

国际信息工程先进技术译丛



Springer

FPGA设计——基于团队的最佳实践

FPGA Design

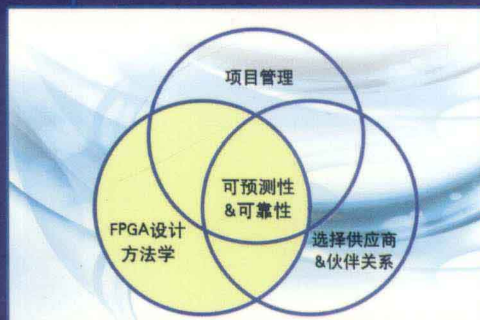
Best Practices for Team-based Design

（美国）Philip Simpson 著

何 春 译

夏宇闻 审





机械工业出版社
CHINA MACHINE PRESS

国际信息工程先进技术译丛

FPGA 设计：基于团队的最佳实践

Philip Simpson 著
何春　译
夏宇闻　审



机 械 工 业 出 版 社

本书根据FPGA设计实践中的经验总结，介绍了一套适用于FPGA设计的最佳实用设计方法学。该方法学涉及了整个FPGA设计流程，从编写设计规范到RTL代码设计，再到设计验证，几乎涵盖了从基本到高级的所有技巧。全书共分为14章，主要包括设计初期的项目管理、设计说明书、FPGA器件选择、团队设计环境，以及设计过程中的电路板布局、功耗和热分析、RTL代码设计、IP及设计重用、硬件到软件的接口、功能验证、时序收敛，设计完成后的在线调试和设计签收等内容，并针对设计中常见的问题和设计优化提供了具体的指导。

本书主要讲述了FPGA设计过程中的经验、方法及技巧，有助于客户解决复杂FPGA设计中的各类问题，对获得高性能设计及缩短设计周期有很大的帮助。

本书可以作为电子工程类、自动控制类、计算机类本科高年级及研究生教学用书，也可供其他工程人员自学与参考。

本书版权登记号：图字01-2012-2202号

图书在版编目（CIP）数据

FPGA设计：基于团队的最佳实践/（美）辛普森（Simpson, P.）著；何春泽．—北京：机械工业出版社，2013.11
（国际信息工程先进技术译丛）
书名原文：FPGA design best practices for team-based design
ISBN 978-7-111-45264-5

Ⅰ.①F…　Ⅱ.①辛…②何…　Ⅲ.①可编程序逻辑器件-系统设计　Ⅳ.①TP332.1

中国版本图书馆CIP数据核字（2013）第310719号

机械工业出版社（北京市百万庄大街22号　邮政编码100037）
策划编辑：林春泉　责任编辑：吕　潇
责任校对：陈秀丽　责任印制：张　楠
北京京丰印刷厂印刷
2014年1月第1版·第1次印刷
169mm×239mm·9.25印张·176千字
0 001—3 000册
标准书号：ISBN 978-7-111-45264-5
定价：49.00元

凡购本书,如有缺页、倒页、脱页，由本社发行部调换

电话服务
社服务中心：（010）88361066
销售一部：（010）68326294
销售二部：（010）88379649
读者购书热线：（010）88379203

网络服务
教材网：http://www.cmpedu.com
机工官网：http://www.cmpbook.com
机工官博：http://weibo.com/cmp1952
封面无防伪标均为盗版

译 者 序

我是电子科技大学的一名青年教师，面对研究生利用 FPGA 做项目过程中的种种困惑，我意识到他们迫切需要在掌握了利用硬件描述语言（HDL）进行编码的基本技巧后，能获得设计方法论方面的指导。这样才能提升设计效率，提高设计系统的性能。我常常在他们提出问题或碰到困难时，将多年来在 FPGA 设计领域积累的经验和体会传授给他们，但终究感觉自己的知识不够系统和全面。

1997 年，我的导师赵和平（航天 501 部）派我到北京航空航天大学，参与到与夏宇闻老师的项目合作中，并顺利地完成了 RS（255，223）编码/解码器设计研究的硕士课题。我在夏老师的具体指导下，掌握了利用 Verilog 设计的基本方法，受益匪浅。在北航学习期间，我了解到夏老师经常阅读国外书籍和文献、学识渊博，认真负责，待人诚恳，乐于提携年轻老师。2012 年初，我向夏老师请教业界有没有关于 FPGA 设计方法学的书时。他立即推荐了 Springer 出版社在 2010 年出版的《FPGA Design：Best Practices for Team-based Design》。

我在大致阅读了这本由 Altera 公司副总裁 Philip Simpson 编写的书后，感觉它几乎涵盖了利用 FPGA 进行系统设计的整个流程，在关键的地方又有所侧重。这本书源于 FPGA 设计过程中的实践经验，介绍的方法学涉及 FPGA 的项目管理、设计规划、资源利用、设计环境、电路板设计、功耗与热分析、RTL 代码设计、IP 和设计重用、软硬件接口、功能验证、时序收敛、在线系统调试和设计交付等多方面的内容。最重要的是这本书全面地介绍了团队合作完成 FPGA 设计的具体方法，对如何组织团队合作进行 FPGA 设计，提出了建设性的意见和建议。这在当今复杂系统的 FPGA 设计中非常实用。书中论述的一些方法，是我不知晓的，是我曾在实践中碰得头破血流才体会到的。越读这本书，越觉得有相见恨晚的感觉。书中所介绍的方法可以供广大 FPGA 设计者、项目管理者学习和借鉴，对缩短 FPGA 项目的设计周期，提高设计性能有很大的帮助。因此，我下决心将这本书翻译出来，把基于团队设计的最佳 FPGA 设计方法学介绍给大家。

夏老师在翻译之初曾给我提了一些专业的建议和指导。当时，我认为已经把握了一些翻译的精髓。然而，随着翻译的进展，遇到的困难远远超过了我的预期。我感觉高质量的翻译工作确实是博大精深的。对于本书而言，如何真实地反映作者的思想，一方面需要有深厚的专业知识，还要能够使用简明扼要、通俗易懂、通顺的文字进行表达。往往一个句子、甚至一个词语的翻译都需要斟酌很久。有时，我觉得翻译得“已经很好”的一段译文，过几天再看，都还有很大

的改进空间。即便如此，每次当我把自己认可的一段译文发给夏老师审阅，夏老师总会提出很多的修改意见。而经夏老师亲手修改后的译文是如此巧妙与贴切，使我感到与夏老师在专业知识、文字功底和翻译水平等方面的巨大差距。

在整个翻译过程中，夏老师所体现出的认真负责的态度让我无比尊敬和钦佩，深深地影响着我。他让我深刻地体会到“翻译无小事”。虽然耗用的时间成倍增加，但是回过头看最后译文的质量有大幅度的提高，十分值得！

在这本书即将出版之际，首先要感谢夏老师在整个翻译过程中给我的具体指导和鼓励，感谢夏老师对每段译文细致负责的修改和审核；还要感谢 Altera 公司上海分公司范名超高级工程师对 13 章译文的极大帮助；感谢北京惠尔科技赵宗明工程师对第 1 章到第 5 章译文所提的修改建议；感谢陈卓立、崔海霞、朱娟、贺江等同学所付出的努力。

一份投入，一份收获。源于实际，用于实际。愿所有认真阅读这本书的人，收获丰厚！

何　春

2013 年 7 月

审校者序

我是北京航空航天大学电子信息工程学院的一名退休教师。在我近50年的工作生涯中，主要工作内容是数字系统的设计。20年前，我已认识到FPGA在数字系统设计方面的巨大潜力，也深刻理解掌握Verilog硬件描述语言是完成复杂数字系统设计的关键。因此，多年来我一直坚持不断地在教学中推广基于Verilog语言的FPGA设计方法学。

2011年底机械出版社的林春泉编审向我咨询有关数字系统设计的新书。我立即推荐了我刚读过的由施普林格出版集团2010年出版的《FPGA Design：Best Practices for Team-based Design》。

该书由Philip Simpson编写。他是Altera公司的资深技术专家，该书源于他多年来为许多FPGA设计客户服务所取得的宝贵经验。书的内容十分广泛，几乎涵盖了FPGA设计工程的所有方面，书中特别强调团队合作完成设计的具体方法，给我留下了极其深刻的印象。书中论述的方法，与我40多年来在工程实践中积累的经验不谋而合，而且总结得十分具体全面。我认为书中所介绍的方法对中国广大的FPGA设计者有很大的参考价值。

不久，电子科技大学的何老师也向我咨询FPGA设计方法学的新书。于是我也向她推荐了该书，并希望她能花点时间把它翻译成通俗易懂的中文。她刚开始有些犹豫，在我答应为她的译稿做修改和审阅后她就同意了，随后我就把她介绍给林编审，并签订了翻译合同。

在整整一年的翻译过程中，何老师经常通过电子邮件与我交流。虽然我用了不少时间修改审核译文，但是由于水平有限，不免存在遗漏和错误，敬请细心的读者不吝指教。

在本书出版之际，让我感谢曾经为本书出版做出过贡献的所有同仁。

夏宇闻

北京航空航天大学电子工程学院退休教授

北京至芯科技公司FPGA设计培训顾问

2013年9月

原 书 序

2006年8月一位工程副总裁，Altera公司的客户之一，找到Altera公司的工程副总裁 米夏·布里奇（Misha Burich），向他咨询怎样才能可靠地预测FPGA系统设计的成本、进度和质量。

那时，我正负责为Altera设计软件制定设计流程的需求，并正为该需求做深入的调研。

当我和那位工程副总裁一起讨论，了解他们的FPGA设计流程中哪些能可靠地运行、哪些不能可靠地运行时，我意识到该问题不只是个别客户所特有的。许多利用FPGA进行设计的公司都存在这个问题。这些公司有一个共同的特点，那就是它们的许多设计团队分布在不同的地点，每个团队擅长完成的FPGA项目各不相同。若将这些设计小组的设计经验整合在一起，其范围十分宽广。然而，设计工具中却没有相应的流程可以让这些设计团队能互相分享设计模块。

在分析了几百位来访客户提供的数据后，我发现在工程团队之间，设计重用的实施十分困难。同时我也注意到，即使在同一公司的不同设计团队之间，甚至同一个设计团队中，各人所使用的设计方法学都有可能不同。

最近，Altera公司已经在自己的FPGA设计软件和IP研发流程中解决了这个问题。

基于Altera公司在帮助许多客户完成FPGA设计时所取得的成功经验和技术，我与Altera公司工程部的顶尖天才们一起研发了这个最佳的实用设计方法学。我把该方法介绍给了一些客户，他们实施后，都取得了巨大的成功。

通过分析过去客户的数据和近三年来客户的反馈，可以越来越清楚地看到设计中团队合作的困难在这个行业中广泛地存在着，它不仅只限于某个FPGA设计公司，而是遍及整个行业。

因此，在过去三年中，我一直在修改和调整FPGA设计的最佳实用设计方法学，试着在几个客户中实施，都取得了巨大的成功。本书全面涵盖了FPGA设计的最佳实用方法学。现在，让我把本书奉献给所有利用FPGA器件实现系统设计的团队。

菲利普·辛普森
美国加州圣何塞

目　　录

第1章　FPGA设计成功的最佳实践

1.1　引言

本书介绍了FPGA设计的成功要诀。这些要诀源于作者在帮助几百位客户解决各自FPGA设计小组所遇到设计难题时的经验总结。在深入了解了他们各自的FPGA设计环境和步骤，哪些管用、哪些不管用的情况之后，作者有能力帮助他们找到在进行系统设计的过程中几个特别需要注意的地方。更重要的是，这些经验促使作者编写一套值得推荐的设计方法学，为设计者解决FPGA设计中的难点提供最佳的具体指导。

本书把阐述的重点放在跨地域的设计小组上。其目的是制订一套必须共同遵守的方法学来协调各设计小组的工作，使设计小组间模块的交流/交易成为可能，从而显著地提高FPGA设计小组的工作效率。

本章为在FPGA上实现系统设计的预期目标设立了路标。

为了获得预期的设计效果，必须遵循以下三个步骤（见图1-1）：

1. 合理的项目规划和全面的审核；

2. 选择恰当的FPGA器件以确保该器件能为项目的今天或未来提供所需的技术；

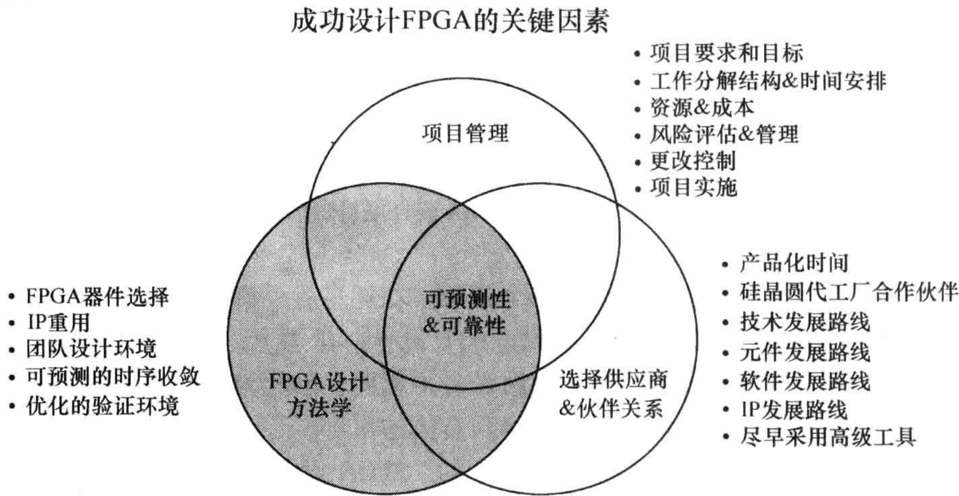


图1-1　FPGA开发三步骤

3. 为了缩短设计周期，确保设计如期完成，遵循并使设计模块可在以后的项目中重复使用，在 FPGA 设计开发过程中，请严格按照本章介绍的方法。

上述三个要素必须协同作用才能保证 FPGA 设计的成功。

应该选择与公司有长期合作关系的厂商开展设计协作。通过共享技术路线图和共同管理现有的项目，不仅可以确保当前项目的成功，也可为今后的项目提供正确的解决方案。协作过程中磨合积累的经验可确保项目的成功。

本书就上述两个议题做了简要的阐述。

第三个议题是 FPGA 设计方法学。本议题是 FPGA 最佳实用设计方法学的主焦点。它涵盖了整个 FPGA 设计流程和从基本到高级的所有技巧。这一套设计方法学独立于 FPGA 供应商，因此这些议题和建议适用于任何 FPGA 器件，确实是最佳实用设计方法。虽然本书中大部分材料是通用的，但也涉及 Altera 设计工具的某些特色，从而使推荐的最佳设计方法更具实用性。

图 1-2 所示的方框结构示意图描绘了最佳实用设计方法的轮廓。

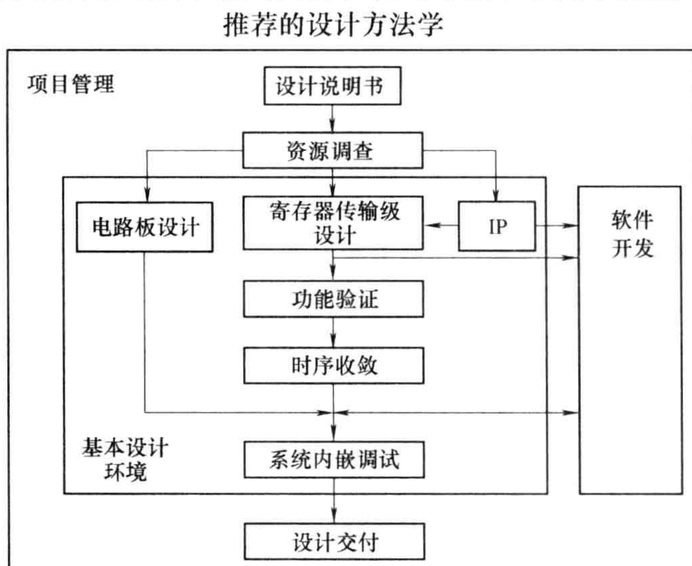


图 1-2　建议采用的 FPGA 设计项目管理方法

图中的每个方框代表本书中的一个章节。功耗另成一章，因为它贯穿设计方法学中的许多领域。关于电路板布局、RTL 设计、IP 重用、功能验证和时序收敛等议题，由于各团队使用的设计方法学各不相同，因此为了达成可靠一致的设计结果，并缩短工期，工程师们需要得到具体的指导。

FPGA 设计所面临的许多挑战并不只是在 FPGA 设计中独有的，它们也是系统设计面临的共同挑战。但与 ASIC 设计相比，FPGA 设计本身确实存在特有的挑战和机遇。随着 FPGA 器件性能的提高，使得面向 FPGA 器件进行更为复杂的

设计成为可能，也促使许多 ASIC 设计师进行 FPGA 设计。这导致了许多设计团队将 ASIC 设计原则照搬到 FPGA 设计上。一般情况下，这对 FPGA 设计流程是有益的；然而它需要与 FPGA 器件带给设计流程的好处相辅相成。FPGA 器件的可编程特性为进行更多的在线系统验证提供了便利。若 FPGA 器件的特性能被正确使用，则可大大加速验证周期，然而当其被滥用时，则会延长设计周期。I/O（输入输出）引脚的可配置性带来了 ASIC 设计中不存在的挑战。FPGA 设计工具与 ASIC 设计工具无论就功能和价格而言，均有很大的差异，但它们都是由 EDA 工业界提供的。

本书致力于采用最佳设计方法以满足设计需求。

建议读者从头到尾认真地阅读本书，当然也可以针对设计流程中的几个块，有重点地阅读本书的个别章节，以应对设计团队面临的最大挑战。

致谢

Misha Burich 为我提供了编写本书（最佳实践方法）的原始想法。Brian Holley 和 Rich Catizone 作为我的客户推进了这个想法，并不断地提供反馈。Chris Balough 鼓励我创作本书。Thomas Sears 允许我接触他的开发团队，没有他的帮助，我不可能写出本书。YK Ning，Jeff Fox，Ajay Jagtiani，Alex Grbic，Joshua Walstrom，Oliver Tan 和 Joshua Fender 为本书（最佳实践方法）提供了原始资料。众多客户为本书做出了很大贡献，因为他们为我提供了 FPGA 设计环境和在 FPGA 上完成系统设计所面临挑战的相关资料。从开始收集数据到写书的整个过程中，我妻子 Jill 和女儿 Kayla，给予了我耐心和支持。

第2章　项目管理

2.1　项目管理的作用

项目管理的目标是期望（设计方）能在经费预算之内，按合同规定的期限和功能完成项目的交付。所以项目管理涉及以下三个方面：

1. 功能；
2. 开发时间；
3. 资源。

项目经理需要合理地平衡上述三个方面，以达到项目目标。

目前，有许多论述项目管理的书籍和培训班，因此本章只对项目管理的要素做一个简要的综述。建议读者参加正式的项目管理培训。

2.1.1　项目管理阶段

每个项目都可划分为三个项目管理阶段：

1. 规划阶段：列出功能清单、制定项目计划以及建立资源库、编制预算；
2. 跟踪阶段：掌握每月的工程进展，进行周计划更新，审核预算、掌握员工状况，以及审核所有工程变更指令单；
3. 收尾阶段：项目回顾、数据挖掘、总结提高，以及下一步的行动计划。

2.1.2　项目持续时间的估算

估算整个项目交付时间（日期）最好能按照以下步骤执行：

1. 在最近成功完成的主要项目中选择一个项目；
2. 创建一个宏模型。为项目规范制定、设计和验证确定主要项目阶段，并得到各阶段确切的持续时间以及交叠时间；
3. 设置整个过程的改进目标。例如完成一个同等难度的项目，希望进度加快10%；
4. 定义项目的复杂度，如设计规格参数和资源利用率。设计规格参数包括设计说明书页数、FPGA资源的大小、RTL代码的行数以及设计性能的技术复杂度；
5. 根据项目的复杂度得到降额因子（derating factor）k；
6. 依据降额因子按比例调整即将开展的项目所需时间；

7. 正确评估项目并做出相应的调整。

2.1.3 计划

项目计划应经常更新，建议至少一周更新一次。

任何项目计划更新会议都应简短，并应将重点放在项目状态信息的收集上。状态信息包括某任务是否已经开始执行、是否已经完成、还需多久才能完成，以及确定其完成进度的所有信息。

项目计划更新会议也应被用来估算某项任务的工期。项目经理必须尊重根据实际资源运作情况而估算出的工期，但也应对任何太离谱的错误估计提出质疑。

2.1.3.1 周计划分析

项目经理需要每周严格分析项目进度。分析过程包括以下 10 个主要任务：

1. 分析和审议关键路径；
2. 重新考虑下周计划的任务；
3. 与评审组的其他人员讨论任务的优先级并最终达成一致；
4. 为加速关键路径的执行制定一个计划；
5. 找出排在关键路径之后有风险的其他路径；
6. 检查分配给关键路径资源上的负荷；
7. 与管理者确认资源的可用性；
8. 确定项目计划中需要更多工作量的部分；
9. 找准活动项；
10. 对任务进行微调。

项目经理一定不要被项目已完成的百分比数所蒙蔽，这一点至关重要。如图 2-1 所示，这是一个非线性函数，所完成的百分比数对于估计剩余任务持续期没有用处（图 2-1）。

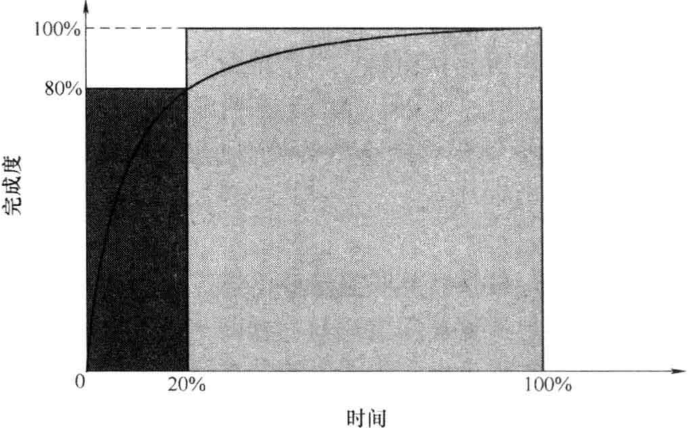


图 2-1 完成百分比情况

2.1.3.2　积极主动的项目管理

按时完成项目交付需要特别积极主动的态度，一定要投入足够的管理力度到项目中去。

由于项目设计环境的不断变化，每周对项目进度表做严谨的更新，管理体制上的这种经常不断地关注不可缺少。

项目的复杂性决定了需要用合适的工具来减轻决策过程的困难。关键路径的确定和管理可简化优先级的设置。

第 3 章　设计说明书

3.1　设计说明书：沟通是成功的关键

在项目正式启动之前，编写一份完整而详细的设计说明书可以避免项目开始时引入的错误，从而可以显著地减少在项目后期修改工程指令单事件的概率。项目后期对设计说明书再做修改将会严重影响项目的进度，增加开发成本，从而显著提高项目成本。设计说明书的任何重大修改都可能导致需要更大规模的 FPGA 器件才能满足设计需求。

编写设计说明书的目的是为了更精确、清晰地传达设计信息。

换一种说法，设计说明书是不同设计小组和人员之间传递信息的一种重要手段。没有一个项目各参与方都认同的、考虑周密的设计说明书，很容易延误项目工期。这是因为在项目后期（的系统整合期间，参与方）各自设想的需求不得不做一些调整；而所有这些都会耽误工期和提高工程造价。上述问题的关键点是“各方都认同”。这就意味着在项目的起始阶段，必须安排一个设计说明书审核步骤。

被各方完全认同的设计说明书确保了参加项目的各小组之间工作的协调和一致性。这也确保了交付的产品符合设计说明书上规定的功能，从而满足客户需求。设计说明书也有助于对项目的开发成本、资源和项目进度进行精确的估算。高质量的设计说明书使我们可以全程跟踪项目的进展，最终制造出高质量的产品。设计说明书也可作为编写产品使用或维护说明书的参考，这些说明书将与产品一起递交给用户。在所有的说明书中，都应对说明书更改的部分做出明确的标记。此外，应该用版本控制软件来保存设计说明书。

FPGA 设计从定义到开发过程的各个不同阶段都必须编写设计说明书。

3.1.1　高级功能说明书

高级功能说明书由系统工程小组创建并负责维护。该文件介绍了 FPGA 设计的基本功能，其中包括 FPGA 和软件接口之间必要的交互，以及印制板上 FPGA 与其他设备之间的接口。高层次功能说明书应由负责 FPGA 设计小组的经理和负责软件设计的经理共同评审。在评审之后，文档应做更新，以反映所做的变更，并回答评审过程中所提出的问题。这一评审过程一直反复进行，直到所有问题都

得到解决，并且整个 FPGA 设计小组都理解并一致赞同这些必要的调整。

编写高层次功能说明书的难点之一是用可理解的文字清晰地描述各项功能。坦白地讲，大多数工程师虽精通数学和科学，但绝不可能成为另一个文学大师约翰·斯坦贝克。

可执行的说明书帮助工程师们解决了这一难题。可执行的说明书描述的是最终系统功能的抽象模型，实质上它是该系统的虚拟原型机。大部分可执行说明书都是用类似“C”（C、C++等）的系统描述语言编写的。这些语言十分适合为想要实现的系统创建功能模型，但它不能描述系统中诸如时序、功耗和设计规模等关键特性。必须再编写一份高层次的说明书作为可执行说明书的附件，才能表示系统的这些关键特性。这个阶段的虚拟原型机是系统模型和测试平台，测试平台也是可执行说明书的一部分。可执行说明书适用于整个开发过程，用来检查具体的执行过程是否能满足可执行说明书的要求。

并非所有的公司都把编写可执行说明书作为 FPGA 设计过程的一部分，但随着越来越多的复杂系统在 FPGA 器件上的实现，可执行说明书的使用也日益普及。

3.1.2 功能设计说明书

FPGA 设计小组应该先编写一份详细的功能设计说明书，它体现的是高层次的功能需求指标。这份说明书应由 FPGA 工程设计小组负责编写，并应经由 FPGA 设计小组和他们的管理者，以及来自系统工程和软件设计小组的代表们评审并通过。最终定稿的说明书应该包括 FPGA 设计的功能、FPGA 与系统其他部分（包括软件）之间的接口细节。

使用这些接口的开发小组必须一致认可与 FPGA 接口的每个细节，这是一项至关重要的工作。

例如在一个设计中，A-D 转换器将采集到的数据输入到 FPGA 中，FPGA 接着将数据传送到微处理器中。该设计拥有软件和硬件接口，因此，这个 FPGA 设计说明书必须包括与 A-D 转换器之间的硬件接口。这个硬件接口必须设计得非常可靠，即便是在极限情况下，也不至于失效。若设计时没有考虑到极限情况，很可能这个接口隐患直到系统测试时才能暴露出来。同时，FPGA 设计说明书还需包含和软件之间的接口。虽然板级测试可以显示 FPGA 将无用的数据送到软件接口，但软件工程师若不知道如何解释或解决这个问题，这将耽误板级测试的进度。在最坏的情况下，可能导致软件或 FPGA 的重新设计，最终将耽误项目的进度。

3.1.2.1 功能说明书大纲

本节将分 8 个部分详细地介绍功能说明书的最低要求。

1. 修订历史记录：这部分内容的样本如图3-1所示。它包含修订日期、作者和修改者，以及已认可的修改要点。

修订历史

版本	作者	日期	更改内容
0.9	psimpson	2009年4月26日	初始版本
1.0	psimpson	2009年5月11日	在数字编解码器中加入详细时序信息
1.1	aclarke	2009年5月30日	2009年5月28日软件工程师审议后进行寄存器映射更改
1.2	jjones	2009年6月3日	添加描述主机处理器接口的部分内容
1.3	psimpson	2009年6月9日	6月4日和软件工程师第二次审议后更新主机处理器接口

图3-1 版本控制页样本

2. 评审会议记录：这部分内容应包括所有审核/讨论设计说明书会议的详情。会议记录应包含会议日期、地点、出席者和备忘录，以及为落实设计说明书而需要执行的各项活动。

3. 目录。

4. 功能概述：这部分内容应描述所属系统的背景。若某项功能属于最终FPGA设计系统的子系统，则应描述它在整个系统中的位置以及目的，即它所能解决的问题。功能概述也应包含必要功能的高层次概述。

5. 参考源：这部分内容应描述功能需求的原由，例如高层次的功能说明书、软件接口功能的需求等。

6. 术语表：这部分内容应包含在文档中用到的所有行业标准术语和缩写语。更重要的是，也应该包含在文档中使用到的公司内部术语。公司内部术语也会引起困惑并造成大量时间的浪费，这是令人吃惊的。许多新员工和来自其他小组的员工在评审会中经常不好意思承认不理解缩写码字的意思，从而引起困惑、耽误决策并经常扼杀创造性。

7. 详细的功能描述：这部分内容是文档中真正有用并值得细嚼的部分。这部分内容应包含对所有使用的算法、设计架构的细节、以及与设计或系统的其他部分之间接口的详细描述。

8. 测试计划：这部分内容应当涉及测试计划，或至少指出编写测试计划的必要性，当测试计划编写完成后，应及时更新该文档。

9. 参考资料：这部分内容应列出为理解该说明书而需要阅读的所有参考文献。

在FPGA设计说明书各部分细节编写的同时，工程小组应该编写许多工程部门内部评审规范，其中包括功能测试计划和质量（QA）测试计划。每个分配到

该项目的工程师都应该各司其责，制定自己的工程计划和功能测试计划。这些个人计划，应先接受审核，然后在整个功能计划中再次复审。这样操作可以确保满足 FPGA 设计的总体需求。

3.1.2.2 测试说明书大纲

1. 修订历史记录：这部分内容的样本如图 3-1 所示。它包括修订日期、作者以及认可的更改。

2. 评审会议记录：这部分内容应包括设计说明书每个评审会的详情。会议记录应包括会议日期、地点、出席者，备忘录和为设计说明书获得批准所需的必要活动。

3. 目录。

4. 测试范围：这部分内容应提供测试计划所覆盖的那些特定功能列表。若测试覆盖范围和任何子系统的测试范围出现重叠，则应详细说明哪些功能是本次测试计划中覆盖的，哪些是参照其他测试计划的。

5. 测试需求：这部分内容应详细列出完成测试所需要的特殊硬件、软件和 EDA 工具。其中还应列出完成测试所需的一切特定的准备工作。

6. 测试策略：这部分内容应包含测试成功或失败的评判标准。本次测试是否需要和其他子系统进行交叉验证？为了满足本次测试计划的要求，现有的测试是否可以直接重用，还是需要改进后才能使用？测试是否可以自动完成？若测试可以自动完成，则应该说明测试是如何自动完成的？如何运行测试？例如，每天晚上能自动开始运行的寄存器自动测试，与在开发板上人工运行，验证屏幕上所显示图形是否正确的人工测试，是两个不同的测试策略。

7. 测试自动化计划：这部分内容描述的是如何使测试自动化，应尽可能地实现测试自动化。

8. 测试的运行：这部分内容应描述预期的测试需要多长时间才能做完？若测试不能自动地进行，则人工完成这些测试需要多长的时间？

9. 测试文档：这部分内容应包含测试案例的描述。按照实际操作标准，测试设计者应使每个测试的激励互相独立，用这样的原则来构造测试平台的基础结构。因此，每个测试案例都应该有自己的测试目录。测试文档应详细介绍如何从回归测试数据库中存取测试结果。当然这是在假设回归测试系统已经建立的情况下。没有建立这样一个回归测试系统的项目注定会失败，因为没有它，检测产品的质量将会非常困难。

对于那些子测试不能自动运行的测试案例而言，其测试文档也应该包含相关的测试步骤。在这种情况下，有必要在文档中说明人工子功能测试是如何进行的。

随着 FPGA 设计工作的开展，应把定期举办设计和验证评审会作为工程实施

过程中程序化工作的一部分，以确保设计工作能完全按计划开展。为了解决项目执行过程出现的实际问题，并消除设计说明书中不太明确的描述，设计修改是不可避免的。评审会为设计的修改提供了沟通的平台。评审会后，设计说明书需要更新和复审。若这些修改与高层次的功能说明书或与 FPGA 的某个接口发生冲突，还应召集相关人员对修改进行正式评审，以落实这些修改。

总之，编写说明书的主要目的就在于设计小组之间的信息交流，以确保设计符合客户方的需求，雇佣足够的工作人员，在规定的期限内按要求完成项目的交付。

功能说明书和测试说明书应按公司内部的标准执行。公司内部标准应与国际标准接轨，并建立在与国际标准（如 ISO9001）兼容的基础上。本书不讨论 ISO9001 标准的任何细节，其具体描述见 http://www. iso. org。

建议进一步阅读：由 Ian Alexander 编写的需求。

第4章 资源调查

4.1 引言

本章分为三个主要部分。

第一部分讲述工程资源，无论使用内部资源还是使用外部合作者的资源都将涉及这部分内容。

第二部分涉及在设计中究竟选择使用公司内部可重用的 IP，还是购买第三方 IP 的问题。

第三部分也是最后一部分，讲解如何选择 FPGA 器件。这部分内容详细介绍了如何选择合适的 FPGA 器件，以使所用器件拥有设计所需的各种资源。这项工作涉及多方面的技术，这些技术可以帮助读者选择合适的 FPGA 器件以保证项目的进度。

4.2 工程资源

项目工程资源的分配属于项目管理任务。每人工作任务的安排是否恰当，人力资源配备是否足够至关重要。从事 FPGA 设计工作，需要考虑的不仅是从事 FPGA 设计的工程师，还必须包括为完成设计任务不可或缺的其他工程师。因此，从硬件工程师的视角而言，应该关注由这些工程师共同组成的整个团队。这个团队中既有 RTL 设计师，也包括熟悉 FPGA 设计软件，并有设计集成经验的工程师，以及具有设计验证经验的工程师。

有些公司，上述这些角色均由同一个人，或由同一组工程师承担。然而，依据设计项目的大小或复杂性，较大的设计项目通常需要由来自不同工程学科，具有不同技术背景的工程师所组成的团队才能承担。从硬件工程师的视角观察，还需要关注电路板设计工作，因此必须确保团队中有电路板布局设计师。他们必须与 FPGA 设计师在工作上保持密切的联系，还需要确保团队人员之间有良好的合作关系。如果电路板上有高速信号，特别当设计中包含高速收发器或高速存储器接口时，团队中还需要有信号完整性工作经验的工程师参与。

如果设计中用到处理器软核，如 Altera 公司 FPGA 中的 Nios Ⅱ 处理器，设计团队中还必须有软件工程师参与。即使只是设计 FPGA 与微处理器的接口，在板

级调试开始时，也希望有软件工程师参与。团队中还可能需要拥有其他专业背景的工程师。例如，某个设计中包含 DSP 算法，而算法设计师本人却不具备硬件设计背景，他自己没有能力在 FPGA 上实现这个设计。所以项目主管必须确保这位算法专家在整个设计过程中和设计实现之后的调试过程中在场，以便随时回答硬件工程师的提问。在做其它高级 IP（诸如 PCIe 或 GigE 这样的主要接口协议 IP）核的设计时，也需要有非硬件专业的有关技术专家在场提供咨询。

确定哪部分工作可利用公司内部的工程资源实现，哪部分工作必须依靠外部专家解决，是工程资源分配中的一个重要决策。

4.3 第三方 IP

设计主管必须关注总共有哪些第三方 IP 可用，其中哪几个 IP 将被用于本设计。同样，他也必须关注公司内部有哪些 IP 可以被重复使用，是否有来自其它项目的 IP 可用于本次使用的 FPGA 系列。假如正在使用第三方 IP，也许还应该关注购买这个 IP 所附带的服务，能否获得相应的咨询服务？对这个 IP 在芯片面积、速度和功能方面完全满足设计需求的信心能达到什么程度？

4.4 FPGA 器件的选择

影响 FPGA 器件选择的七个主要因素排列如下：

1. 某款器件所具有的特色。为了满足设计中的某些特殊需求，设计师不得不选用某一款 FPGA 器件，因为其他各款 FPGA 器件均不能满足这些特殊需求。

2. 器件规模（密度）。设计总共需要多少个逻辑单元？如何用逻辑单元和存储块构造应用所需的专用乘法器模块？构造的优劣将对所需 FPGA 器件的价格产生很大的影响。

3. 速度需求。这将对选择 FPGA 器件的系列和必须选用的速度等级产生影响。同样，这也会对 FPGA 器件的价格产生很大的影响。

4. 器件引脚。需要什么类型的封装？封装类型和设计中输入/输出（I/O）引脚的数量将影响 FPGA 器件的成本和电路板设计。封装类型也将影响设计中输入/输出的信号完整性和性能。

5. 功耗。允许设计消耗功率的上限是多少？选用哪种 FPGA 器件可以把功耗控制在预算的范围内？

6. IP 的可获得性。需要的 IP 是否容易到手。

7. FPGA 器件的可获得性。确保需要时可以随手得到想要的 FPGA 器件。

以上七个方面需要我们更为详细的关注。

4.4.1 FPGA 器件的特殊功能

首先应该关注的方面是 FPGA 器件的专用资源。设计需要高速串行接口吗？如果需要的话，需要多少个通道，各通道的性能又如何？许多 FPGA 器件都自带收发器。收发器性能可分为三个档次，其最高速度分别达到 3.125Gbit/s，6.5Gbit/s 和 10Gbit/s +。这些影响到设计性能和 FPGA 成本，因此它们是器件选择过程中的重要因素。还需要关注设计的带宽要求。收发器的速度和数量决定了带宽。例如，在通信市场中，如果要实现 100 千兆以太网，可能至少需要 10 个 10Gbit/s 的收发器通道。

同样，如果正在实现的算法是运算量密集的算法，诸如 DSP 加密算法或雷达应用方面的算法，这就要求 FPGA 器件具有大量的 DSP 模块，并拥有足够多的 RAM 模块可与这些 DSP 模块接口。DSP 模块的配置也十分重要。存储模块的深度和数量会影响有多少处理可以在芯片上执行，如果内部存储器不够多的话，就不得不使用外部存储器。内部存储器在 DSP 运算中用于缓存算法处理各阶段间的处理结果，是非常重要的。还需要关注专用 DSP 模块的数量和配置，关注需要执行的乘法运算的位宽是多少。如果 DSP 模块没有足够多的位宽，就不得不用逻辑来组合 DSP 块以实现所需功能。这会影响正在执行操作的性能。

需要有多少个内部 RAM 模块呢？当考虑使用处理器软核做设计时，这个问题变得越来越重要。使用内存块作为缓存可以显著提高处理器软核的性能。可用 RAM 模块的大小也十分重要。若设计中要用到大量的 FIFO，则需要特别注意可用 RAM 模块的个数，而对每个 RAM 模块中可用位数的需求可以放松点。使用存储模块来实现 FIFO，很可能造成内存位的浪费，这是众所周知的缺点。

设计调试过程中的资源消耗也需要考虑。在设计调试期间，内部存储器块常被用于保存来自嵌入式逻辑分析仪的数据。

4.4.2 FPGA 的规模选型（密度）

在进行 FPGA 器件规模选型时，通常设计尚未完成，也不完善，因此很难确定所需要 FPGA 器件的规模，这时往往需要根据以前的经验来选择 FPGA 器件的规模。许多设计是基于以往的设计，这对 FPGA 器件的规模选型会有帮助。通常的做法是针对本次设计想要用的 FPGA 器件系列，重新编译旧版本的完整设计，或者编译本次设计中所用到的那部分旧代码，以便获得一个大致正确的规模估计。如果设计中将使用 IP，就应编译该 IP，并将结果添加到在总面积估算中，如果正准备选用第三方供应商 IP，就应从供应商那里得到其面积估计。总之，如果存在老版本设计的话就应以它为基础，并把将使用的 IP 占用的面积估算在内，然后根据经验，考虑为实现新功能还需要增加多少额外的面积。一旦完成了

这项工作，在已有的面积预估上再增加25%。在FPGA器件规模选型时，应该选择一个比预想需求大一些的FPGA器件，这就是面积预估方程中增加25%的出处。

设计师应该选择一个比预期需求规模更大一些的FPGA器件。由于设计规模有不断扩大的趋势，这样做可以确保选定的器件可容纳下设计的发展规模，并且能使时序收敛。设计师就不需要在器件利用率已达到95%时，还在为时序收敛而绞尽脑汁，或为了使设计恰好能被选定器件容纳下，而不得不砍掉系统的某些功能。

使用较大器件的另一个好处是有助于在FPGA系统芯片内部加快对设计的检查。如果FPGA器件有余量的话，布局和布线软件可能不必费大力就可以满足时序需求，从而缩短编译时间。这对硬件工程师和软件工程师都十分有利。越早拥有可实际使用的器件，软件工程师就可以越早在目标硬件上试着运行自己编写的代码，从而加速代码的开发进程。这样做，使设计师能够在设计周期的起始阶段就着手硬件和软件的调试工作。

若FPGA留有余量，则由设计后期修改或产品版本更新所增加的逻辑单元就能比较容易地被其容纳。这是FPGA器件留有余量的又一好处。

设计在FPGA器件上正常运行后，如果FPGA器件上还有大量未使用的资源，可以换一个较小的FPGA器件以降低成本，而不必担心影响项目的进度。一些FPGA供应商提供的设计工具拥有在同系列的不同规模（密度）器件之间自动完成设计移植的功能，同时能保持引脚不变。但这些功能对所使用的引脚（I/O）资源是有限制的，只能使用同系列特定规模（密度）范畴内器件上都存在的引脚（I/O）资源；这样做的好处在于能改变设计，使之适用于较大规模或者较小规模的FPGA器件，可避免电路板的重新设计。如果这一功能在所选择的FPGA供应商提供的工具软件清单上没有，可以通过参考器件手册和操作说明书自己编写引脚移植程序。虽然人工编写移植程序的过程非常痛苦而且容易出错误，但在没有自动化流程的情形下，这个投入是十分值得的。（译者注：用纯手工方式逐个定义引脚是不现实的。）

关键点在于必须保证设计师选用的FPGA器件系列，具有允许在不同规模的FPGA器件之间进行设计移植并保持引脚不变的能力。

建议选择一个FPGA器件，它在规模（密度）上既能向上兼容以适应未来设计的增长，又能向下兼容以便尽可能地降低成本。

如果打算推出价位不同、功能不同的多档次产品，上述功能非常有用。只需要设计一种电路板，逻辑设计也只需做一次，就能实现产品的系列化。若市场需要较低价位的产品，从FPGA中去除一些功能即可。通常不同档次产品里的电路板和安装的FPGA完全相同，所不同的只是FPGA的编程文件，低档次产品的

FPGA 编程文件中的功能被减少了一些而已。只要保持引脚兼容，就可以去除一些功能，以便用规模小一些的 FPGA 器件实现设计，从而进一步降低材料的成本。

4.4.3 速度需求

可以根据以往的设计经验来确定设计的速度需求。对早就拥有的设计或对设计进行编译可得到设计在目标器件上实现后的速度性能指标。用这个好办法可以估计出其他设计块在目标器件上实现后的性能指标。

FPGA 供应商提供的产品说明书也是关于设计实现后性能指标的优质信息源。这些产品说明书告诉设计师在时钟和输入/输出（I/O）性能方面期望能够达到的绝对最大值。虽然这些指标是可以达到的，但真正要达到这些指标可能会增加布局布线的难度，延长时序收敛周期，因此应将速度指标降低约 15%，给时序收敛留出安全余量。

速度等级的选择将直接影响 FPGA 器件的价格。选用 FPGA 时，建议从一开始就选用速度等级最高的器件，尽快拿到装有该 FPGA 器件的电路板，从而及早进行软件调试和硬件功能检查。若用速度等级最高的 FPGA 器件实现设计，则比较容易满足设计的时序需求，布局布线工具不难达到时序收敛的设计目标，好处是设计编译工作可以很快地顺利完成。设计后期，当设计功能接近完善之际，可以选用一个较慢的 FPGA 器件来实现设计，以降低产品的成本。

4.4.4 引脚

设计需要的输入/输出（I/O）接口类型将影响 FPGA 器件所需要的引脚数目和封装类型。必须知道所需的 I/O 标准和对 I/O 驱动强度的要求。需要多少个引脚？电源供电要求是什么？在没有开始设计时，要确定这些要求是好弄清楚所使用的 FPGA 器件是如何与电路板接口的。同时也需要关注设计中的信号完整性问题。设计中是否存在大量引脚接口可能同时触发的情况，如果有的话，设计是否存在同时开关噪声（Simultaneously Switching Noise，SSN）问题？值得注意的是，键合（wirebond）封装的 FPGA 器件通常比倒装（flip chip）的 FPGA 器件具有更差的信号完整性和 I/O 性能。

建议在考虑设计的引脚数目时，为系统内嵌调试预留一些引脚。至少应预留 15% 的 FPGA 器件引脚，用来将内部信号引到芯片外供逻辑分析仪进行分析。

4.4.5 功耗

根据设计说明书可以知道设计的功耗预算。FPGA 器件需要多少种电源？大多数现代 FPGA 器件要求有多个电源供电，为 IP 核、I/O、通常还有收发器提供

各自独立的电源层。FPGA 器件需要的电源个数越多，电路板上元器件成本就越高，电路板设计的复杂程度就越高。

同样，之前 FPGA 的设计经验可在本次设计的功耗估算中发挥作用。本书的第 7 章将专门介绍功耗估计，这将有助于克服这一难题。

总之，建议根据 FPGA 供应商提供的功耗评估软件和以往的经验共同确定设计将要消耗的功耗。

4.4.6 IP 的可用性

在某一 FPGA 器件系列上已有的 IP 可能在准备选用的这种 FPGA 器件系列中还没有被引入或没有被验证。这是新投入市场的器件常有的情况。当 FPGA 器件的硅工艺只有不到 6 个月时间时，IP 接口往往是麻烦。这些 FPGA 器件通常没有完全定型，只有一个初步的时序模型。只有当时序模型最终确定的时候，高性能 IP 接口的时序收敛才能得到保证。

4.4.7 器件的可用性

如果项目涉及尖端技术，有可能会考虑使用市面上最新的 FPGA 器件。最新的 FPGA 器件在将来价格会更合理一些。如果设计将在 12 个月之内投产，将在 5 年后批量生产。那么这样在进行批量生产时正逢所使用的 FPGA 器件工艺成熟且其价格最低的时候。在这种情况下，决定使用市面上最新的 FPGA 器件在经济上是划算的。

4.4.8 小结

为了快速有效地将 FPGA 器件和虚拟设计融合在一起，实话实说，我们建议设计师在选择 FPGA 器件时，启动以上成功的 FPGA 器件选择流程。这样，设计师将对器件需要什么类型的接口有一个清晰的认识，也有助于确定引脚要求，从而简化 I/O 规划。通过创建虚拟设计，设计师将对 FPGA 器件内部资源的利用有一个总体概念。这一方面可以为达到设计的性能指标提供指导，还能使设计师在设计早期就进行功耗评估。虚拟设计应包括设计中将用到的每一个 IP 模块，它的创建有助于设计师选用合适的 FPGA 器件。

第5章　设计环境

5.1　引言

为了顺利地完成 FPGA 系统的设计，有必要把所有的设计工具、技术和设备组合在一起形成一个良好的氛围，这就是最佳的 FPGA 设计环境。每个公司的 FPGA 设计环境通常有自己的特色，可以满足该公司的特殊需求。然而，不同公司的 FPGA 设计环境之间也存在一些贯穿整个设计过程的共同要素。本章的目的就是让您了解按进度计划顺利完成 FPGA 设计项目对设计环境的最低要求。设计环境可表示为以下五个要素：

1. 脚本化环境；
2. 与版本控制软件之间的交互；
3. 问题跟踪系统的使用；
4. 回归测试系统；
5. 为分析而进行的数据收集。

5.2　脚本化的环境

FPGA 设计工程师面临的挑战之一是在 FPGA 设计过程中，什么时候应该使用脚本化的设计流程，而什么时候应该使用图形界面（GUI）设计环境？

脚本化的设计流程适用于以下五种情况：

1. 项目的创建；
2. 编写设计任务分工的说明书；
3. 设计的编译，尤其当设计者使用计算机集群环境进行编译时更是如此，因为使用脚本化编译环境，允许设计者通过服务器，对分布在集群环境中的编译文件进行批处理；
4. 功能验证和回归测试；
5. 与版本控制软件的结合。

以上这五个方面涵盖了 FPGA 设计的大部分流程。这似乎是在说设计流程的每一阶段都值得推荐使用脚本，但这个观点并不完全正确。对任何重复的任务，确实应该使用脚本。使用脚本有助于其他用户较容易地重现原设计者的环境和结

果。

那么，建议在什么时候使用图形界面呢？

图形界面应该用于设计流程中的交互部分。也就是那些需要根据得到的结果而改变操作的地方。例如以下的三种情况：

1. 设计的系统内部调试。

2. 布局操作。关注布局的细节可对器件的架构和可用资源有更好地了解。布局操作还可以在基于团队的设计环境的布局中，为用户的设计创建一个物理布局。

3. 开始使用新工具时。图形界面为建立第一个项目和展示工具的特性及功能提供了一个很好的方式。一旦熟练掌握了这个工具，建议将设计转移到脚本化的设计环境中。

脚本程序的使用能在重复的任务上节省时间和精力。它最大的益处之一是在基于团队的设计中简化团队各成员间的任务交接。如果某人从另一个工程师那里接手一个项目或设计模块；该工程师不必为了得到设计结果而编写详细的说明来描述需要做些什么，只需要给他们脚本就行了，因为脚本本身就是文档。新接手的工程师看懂脚本之后运行脚本即可。他们就可以从上一个工程师停止的地方开始新的工作了。EDA 工具作为 FPGA 设计流程一部分，几乎都有脚本接口：包括创建批处理文件的命令行接口和在项目中创建配置的分配脚本。EDA 工业界大多数供应商已将 TCL 作为编写其 EDA 工具接口的标准语言，允许 TCL 脚本调用其各种工具。

5.3 与版本控制软件的交互

版本控制软件可为设计的修改提供历史记录。当正在进行某项 FPGA 设计时，有必要知道在版本控制软件中需要递交（check in）和取出（check out）文件的最小集合。应该尽量减少这些文件的数量，因为递交的文件越多，所需的存储空间就越大，操作就越复杂。设计的每次修改，都需要将整个 FPGA 项目递交到版本控制软件中。而一个好的脚本环境有助于简化这一过程。脚本的初始设置和所需递交、取出文件操作和确认操作过程可能十分复杂。然而，这些脚本一旦建立后，就可以与专注该项目开发的工程师们共享。如果能用脚本重建项目或描述项目，与版本控制软件间的交互将会变得简单得多。

为重现结果，不同的 FPGA 设计工具要求把不同的文件集置于版本控制软件中；因此，当使用不同 FPGA 厂商提供的设计工具时，版本控制软件的设置可能显著不同。而原则却都是一样的。若 EDA 工具使用文本文件，则它与版本控制系统之间的交互将变得十分简单。若 EDA 工具使用二进制文件保存关键信息，

则它与版本控制系统的交互将变得比较复杂。

为了使设计师能重现以前的编译结果，究竟需要把哪些文件递交到版本控制软件中去处理呢？直到今天，有关这项商业机密的公开发布工作，FPGA 厂商仍旧做得很差。如果在 FPGA 设计流程中使用了多种工具，这个过程将变得更加复杂。为了理解他们推荐的好办法，建议设计师与各工具的供应商接洽。

设计环境中所使用的目录结构对如何使用版本控制软件而言至关重要。设计环境的目录结构中包括了 RTL 设计文件的位置、RTL 库和 IP 库的位置、若使用软处理器还有 C 代码和程序存储区的位置、仿真测试平台的位置、寄存器测试结果的储存位置，以及在 FPGA 软件或其他 EDA 软件中用来编译设计的脚本文件所在的位置。为了能使用这些文件的正确版本，必须将所有这些要素顺利地连接在一起。

在实验室中调试设计时应该避免发生以下这些情况：在 FPGA 上使用了错误的固件代码、在软处理器上加载了旧的源代码、设计师修改的 RTL 文件的版本已经过时。正确使用版本控制软件提供了一个可以防止类似情况的发生。因为报告文件能说明设计的状态，所以应该把报告文件也保存在版本控制软件中，这将为从事同一项目开发的其他设计师们提供有价值的信息。

5.4　问题跟踪系统的使用

问题跟踪系统不列在 FPGA 厂商的供货清单上。但是，能肯定的是问题跟踪系统是一个工具，FPGA 厂商使用它作为他们的工程和产品规划过程的一部分。为了满足各个公司的需要，问题跟踪系统往往在公司内部自主开发。实际上许多 EDA 工具供应商和 FPGA 厂商都有一个用户到它们系统的接口，用于提交问题报告。

市场上可以购买到商业化的问题跟踪系统。这些问题跟踪系统本质上是数据库系统，只是为了满足不同公司的需要而添置了一个可定制的前端。在您的设计环境中，您将使用该跟踪系统去跟踪 FPGA 设计中所有已知的问题。这使设计工程师们能在设计出现问题时把它们记录下来。问题跟踪系统为设计团队提供设计的即时状态，并能在整个设计过程中用来跟踪设计的稳定性。它使得团队中其他成员可以知道您设计中所存在的问题，从而避免发生以下情况，即他们费力地调试自己那部分系统出现的一个问题，而这个问题是由您的设计而引起的。通过研究这些资料，可以决定是否使用由于这个问题而导致的特殊情形，还是恢复到原先这个问题还没有显现时的情形。

问题跟踪系统还允许用户详细记录问题是如何解决的。这使得团队成员之间互相合作一起解决设计中出现的问题非常方便。这对跨越多个时区基于团队的设

计环境非常有帮助。

如上所述，我们可以用问题跟踪系统给项目的健康状况拍快照。为了做到这一点，问题跟踪系统必须与回归测试系统相结合，这样每次回归测试失败后，将对照被测试的情形把问题跟踪系统中的问题报告自动归档。

5.5 回归测试系统

作为测试的一部分，设计工程师需要创建测试点以证明自己的设计符合功能要求。想要为设计提供健康检测，必须拥有一组经常在设计上运行的测试。这些测试使您拥有这样的信心，即在今后的设计修改中，不会再引入以前出现过的问题，也不会破坏现有的功能。在第 11 章中我们将会更详细地讨论回归测试问题。

5.6 何时升级 FPGA 设计工具的版本

FPGA 厂商通常每年至少发布两次有较多修改的新版本，并附带一组补丁包，其中包含程序错误修复和时序模型修正等补丁。如果有一项持续时间超过 6 个月的设计，项目管理者不能不回答这样的问题：何时采用 FPGA 设计环境中最近发布的设计工具新版本？什么时候却应该保持您正在使用的设计工具的版本不变？

这个问题的回答取决于当 FPGA 设计工具版本更新时，设计正处于流程中的哪个阶段。如果正处于设计的早期阶段，就应随时升级到 FPGA 设计软件的最新版本，除非知道新软件有严重的问题。这样可以在新版本软件中获得最新的错误修复和最新的功能。大多数 FPGA 设计软件的新版本通常在一定程度上会缩短编译时间。

若设计即将完工，而且您正在使用的这个 FPGA 厂商所提供的软件版本含有目标器件最终的时序模型，则应考虑保持正在使用的设计软件版本不变。有一个例外就是若正巧遇到设计软件中的一个错误，它影响了设计进度，则为了修复这个错误，可能不得不升级设计工具。

如果您的设计即将完工，但 FPGA 厂商提供的仍是初级的基本时序模型，一旦有了最终时序模型，设计软件必须升级到新版本。设计软件的升级可能会遇到问题，因为供应商提供的 IP 模块很可能也要跟着升级，这将增加许多工作量，尤其在设计验证过程中更是如此。强烈建议对照产品或最终版本的 FPGA 时序模型进行设计验证。

有些 FPGA 厂商提供的设计工具可以用高版本软件读取低版本软件中的设计数据库。因此，为验证在改用新发布的最终时序模型的情况下，设计是否仍然能

满足时序需求，并不需要重新编译设计，只需重新运行时序分析即可。

5.7 FPGA 设计环境中常用的工具

FPGA 设计软件：FPGA 设计软件来自 FPGA 厂商，其中包括 FPGA 布局布线软件和时序分析工具。主要的 FPGA 厂商也提供 RTL 综合工具、高级时序收敛工具、片上调试工具，以及布局工具。

FPGA 综合软件：FPGA 综合软件可能来自 FPGA 厂商，也可能来自 EDA 综合工具供应商，诸如新思科技公司（Synopsys）或明导国际（ Mentor Graphics）。大多数 FPGA 综合工具支持 Verilog 和 VHDL，现在一些 FPGA 综合工具也支持 System Verilog。

仿真工具：有些 FPGA 厂商提供仿真工具，但至今为止所使用的仿真工具中的大多数来自 EDA 工具供应商。其中最流行的工具是 Mentor 的 Modelsim 和 Questasim、Synopsys 的 VCS、Cadence 的 Incisive，以及 Aldec 的 Active HDL 和 Riviera Pro。有些工具包含了一些高级功能，诸如基于验证的断言、跨时钟域的检测等。

形式验证工具：在 FPGA 设计过程中通常不使用形式验证工具，这是因为这类工具在执行优化时所设置的一些限制。为了顺利地完成验证，在使用这些工具时，可以对其执行优化。

时序分析工具：EDA 工具供应商提供了几种时序分析工具。然而，由于 FPGA 厂商也提供自己的时序分析工具，所以 EDA 工具供应商提供的时序分析工具很少用于 FPGA 设计流程中。建议使用 FPGA 厂商的时序分析工具进行 FPGA 的时序分析，这是因为用做时序签收的时序约束，还可以用于布局布线的优化。

建议在 FPGA 验证中不要使用 EDA 时序分析工具，而在电路板的时序分析中要使用 EDA 时序分析工具。

电路板设计工具：EDA 工具用于电路板设计中。它们包括电路板原理图工具、电路板布局布线工具和信号完整性工具。在信号完整性工具中使用的 HSPICE 和 IBIS 模型来自 FPGA 厂商。

高层次综合：这个领域的大多数工具都是处理用“C”语言或“C ++”语言编写的设计模型，综合后产生与之对应 RTL 级代码或 FPGA 网表。这些工具在 FPGA 市场上被接受的进程一直很缓慢。这些工具已经成熟了许多，它们在某些应用领域的设计中正在缓慢地取得进展，而在创建完整的 FPGA 设计方面，无显著进展。这些工具主要集中在高性能计算领域和 DSP 算法的实现领域。

所有这些产品均由 EDA 公司提供。

下一类型的高层次综合工具是基于模型的设计工具。这些基于模型的高层次

综合工具使用 Mathworks Simulink 环境中的优化库，它们的目标市场是军方和调制解调器设计。这些工具依赖于 Mathworks Matlab 环境。可以从主要的 FPGA 厂商和 EDA 公司购买到这些高层次综合工具。

负载分配软件：该软件用于调度正在计算机集群中处理的作业。负载调度分配的解决方案在 FPGA 开发案例中经常使用，尤其在基于脚本的设计流程中更是如此。负载分配软件有商用软件包，也有免费软件。FPGA 软件中的一些选项包含了某些形式的负载分配软件。

版本控制软件：版本控制工具虽不属于 EDA 工具范畴，但却是设计流程中相当重要的环节。FPGA 设计中常用的版本控制软件有 Clearcase、Perforce 和 PVCS。

第6章　电路板设计

6.1　FPGA 器件给电路板设计带来的挑战

为了满足当今系统设计高性能和高带宽要求，FPGA 器件提供了大量的引脚，并且引脚的开关速度越来越快。封装引脚数的增加，加上器件支持许多不同的 I/O 标准和封装类型，这个实际情况给顺利、高效、正确地创建 FPGA 器件的引脚带来了巨大的困难。修改引脚分配会导致电路板的多次改版，从而造成成本的增加，耽误项目的进度，其代价十分昂贵。

FPGA 器件可提供灵活的引脚，并支持多种不同类型的 I/O 标准，允许用户对引脚的驱动能力和翻转率加以控制。这一灵活性导致了合理分配 FPGA 引脚的复杂规则，从而对 FPGA 与印制电路板（Printed Circuit Board，PCB）的连接设计提出了新的要求。

由于 FPGA 封装的引脚数十分庞大，从而造成 EDA 工具流程中 PCB 设计软件和 FPGA 设计软件之间数据管理方面的困难。

由于高性能 PCB 设计的复杂性，所以 PCB 设计必须在系统设计的早期就开始执行。这造成 FPGA 的最终引脚与 PCB 设计周期协调一致的困难。通常电路板要早于 FPGA 设计完成其布局布线。但实际上，目前 FPGA 和 PCB 设计同时开工正变得越来越普遍，而对许多系统设计师而言，PCB 的设计经常在 FPGA 的 RTL 级代码存在之前就已经完成了！

在设计周期的初期，项目所需的 FPGA 器件规模的大小很难预测。针对这个问题，大多数 FPGA 器件系列都有其技术解决方案，即支持封装相同、规格不同的 FPGA 器件之间的引脚兼容。因此，建议设计者在封装相同、有多种规格的某个系列中选择合适的 FPGA 器件。对 PCB 设计师而言，在不同规格 FPGA 的每种器件中达到引脚兼容十分困难。然而，上一章已经提到，有些 FPGA 设计工具通过一个通常称为器件移植的功能，可以方便地为用户提供帮助。器件移植是将设计从一个 FPGA 器件转移到同一系列中某个封装相同、规格不同的 FPGA 器件上的能力。在保持电路板布局布线和引脚分配不变的前提下，器件移植能使设计从其原本的目标器件转移到另一个引脚相同，而规格不同的器件上。当选择 FPGA 器件时，用户可在 FPGA 厂商提供的软件中选择移植功能。该功能可以防止用户在分配引脚时误用同一系列不同规格器件之间不兼容的引脚。由于 FPGA 设计中

很可能发生无法预知的变化，为了保险起见，建议将引脚兼容要求纳入设计规划，特别是在FPGA设计早期确定引脚的时候更要注意。当设计需要修改而导致逻辑规模显著增大时，引脚兼容技术使设计换用规格大一些的FPGA器件成为可能，或者只要设计的规模允许，还可以换用规格小一些、价格也便宜一些的FPGA器件。

系统性能的提升和带宽的增大对引脚的速率提出了更高的要求。FPGA器件能以533MHz的高速率，通过64bits的DDR 3 SRAM接口向SRAM写入数据。每个引脚的数据率高达1067Mbit/s。这将使FPGA器件大量的引脚同时发生翻转，引脚同时翻转所产生的噪声又会导致器件功能失效。因此，FPGA器件必须拥有能避免同时翻转噪声（SSN）的引脚，而且FPGA器件的引脚与电路板连接时也必须避免SSN问题。

许多FPGA器件还包含速率高达11.3Gbit/s的收发模块，并且支持多种I/O协议，例如PCI EXPRESS，Serial RapidIO®，Gigabit Ethernet（GbE）等。基于高速收发器的接口与电路板连接时必须小心谨慎，以避免信号完整性（Signal Integrity，SI）问题。

现在我们已经认识到设计一块包含FPGA器件的高性能系统PCB所潜在的风险，因而将重点放在那些可用来确保PCB设计一次成功的技术上。本章其余部分将详细地描述高速率PCB设计所面临的困难；介绍PCB设计过程中的小组分工。本章其余部分还提出一套应对所有这些困难的方法学，最后总结出一套可用于任何FPGA项目使FPGA的引脚分配和PCB设计获得圆满成功的要诀。

6.2　工程师的角色和职责

参与包含FPGA器件的系统电路板设计的工程师，可以按专业分成三个不同的小组，即FPGA组、PCB组和信号完整性组。在某些机构中，他们的功能有交集，但总体上，由于专业学科不同，不同工程师或者工程师小组所履行的职责也不同。

6.2.1　FPGA工程师

FPGA工程师们熟悉FPGA设计软件。这些FPGA工程师通常负责编写和验证设计的RTL代码；也负责在FPGA中实现设计，并在最终系统中帮助调试设计。

在PCB设计中，FPGA工程师继续发挥着作用。他负责从FPGA设计软件中生成FPGA的引脚分配图。因此，FPGA工程师应与PCB工程师密切配合，根据PCB设计工程师提出的建议，及时修改FPGA设计，更新引脚分配，并验证所做

的改动。

FPGA 工程师还要与信号完整性工程师密切配合，为他提供由 FPGA 设计软件生成的 FPGA 引脚分配图、HSPICE 模型、IBIS 模型，以及网表。

6.2.2 PCB 设计工程师

PCB 设计工程师熟悉 PCB 原理图软件和布局布线软件，通常负责创建电路板原理图，其中包括器件符号的生成。同时他还负责电路板的布局布线。电路板的布局，特别是电路板的布线基本上取决于电路板上元器件的引脚分配。由于 FPGA 的引脚分配会严重影响 PCB 设计工程师的工作进度，同时也潜在影响电路板的成本，因此 PCB 设计工程师在很大程度上会去影响 FPGA 的引脚分配。虽然 PCB 设计工程师对 FPGA 器件的引脚分配会产生很大的影响，但他通常没有使用 FPGA 设计软件的想法。这使得在 FPGA 设计师和 PCB 设计师之间必须建立传递信息的有效手段。这实际上是一个双向的需求机制，即从 EDA 工具厂商那里购买的 FPGA 设计软件和电路原理图设计软件之间的双向接口机制。如今，有些 EDA 工具厂商提供了与 FPGA 设计软件的双向接口。然而，在这两类工程师之间进行信息交流的最常用的接口却是 Microsoft Excel。大多数 FPGA 厂商提供的 FPGA 设计软件具有读写 .csv 格式文件的能力，可以与 Microsoft Excel 接口。同样，有些原理图软件包也能读 .csv 格式文件。工业界电路板设计师的普遍做法是创建一个脚本，从 .csv 格式文件或者 FPGA 软件的引脚报告中生成相应的原理图符号。因此，.csv 格式文件有多种用途，下面列出了其中两种：

1. 该文件可用于把 FPGA 设计文件和它的电路板设计文件合并为一个整体。

2. 该文件可用于说明引脚的分配。因此，它应当被保存在版本控制软件中。

下面列出了一个 .csv 文件示例，该示例文件用于说明 FPGA 设计软件和电路板原理图软件间的接口，详例如图 6-1 所示。

其要点是 .csv 文件包含了 I/O 标准和电流强度的许多细节，它比引脚分配文件要详细得多。I/O 标准和电流强度的细节会影响电路板上的信号质量，也会影响 I/O 的时序。

PCB 设计工程师必须与信号完整性设计工程师沟通，为他们提供电路板布局的具体细节，以便信号完整性设计工程师建立该电路板的信号完整性模型。

6.2.3 信号完整性设计工程师

信号完整性（SI）设计工程师应该熟悉来自领先的 EDA 厂商（例如新思科技公司（Synopsys）、明导国际（Mentor Graphics）、安捷伦（Agilent）等）的信

号完整性仿真软件。他们负责验证信号的质量（例如过冲/下冲），并将同步翻转噪声（SSN）控制在规定的范围之内。总而言之，SI 工程师负责验证电路板的时序是否达到系统的要求。

Pin Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Current Strength
clk_in	Input	PIN_B13		4B4_N1	3.3-V LVTTL(default)	24mA(default)
in_port_to_the_button_pio[3]	Input	PIN_AE6		8B8_N1	3.3-V LVTTL(default)	24mA(default)
in_port_to_the_button_pio[2]	Input	PIN_AB10		8B8_N1	3.3-V LVTTL(default)	24mA(default)
in_port_to_the_button_pio[1]	Input	PIN_AA10		8B8_N1	3.3-V LVTTL(default)	24mA(default)
in_port_to_the_button_pio[0]	Input	PIN_Y11		8B8_N1	3.3-V LVTTL(default)	24mA(default)
ext_flash_enet_bus_data[7]	Bidir	PIN_A8		3B3_N0	3.3-V LVTTL(default)	24mA(default)
ext_flash_enet_bus_data[6]	Bidir	PIN_B8		3B3_N0	3.3-V LVTTL(default)	24mA(default)
ext_flash_enet_bus_data[5]	Bidir	PIN_C8		3B3_N1	3.3-V LVTTL(default)	24mA(default)
ext_flash_enet_bus_data[4]	Bidir	PIN_D9		3B3_N1	3.3-V LVTTL(default)	24mA(default)
ext_flash_enet_bus_data[3]	Bidir	PIN_G10		3B3_N1	3.3-V LVTTL(default)	24mA(default)
ext_flash_enet_bus_data[2]	Bidir	PIN_F10		3B3_N1	3.3-V LVTTL(default)	24mA(default)
ext_flash_enet_bus_data[1]	Bidir	PIN_C8		3B3_N1	3.3-V LVTTL(default)	24mA(default)
ext_flash_enet_bus_data[0]	Bidir	PIN_D8		3B3_N1	3.3-V LVTTL(default)	24mA(default)
out_port_from_the_led_pio[7]	Output	PIN_AA11		8B8_N1	1.8V	12mA(default)
out_port_from_the_led_pio[6]	Output	PIN_AF7		8B8_N1	1.8V	12mA(default)
out_port_from_the_led_pio[5]	Output	PIN_AE7		8B8_N1	1.8V	12mA(default)
out_port_from_the_led_pio[4]	Output	PIN_AF8		8B8_N0	1.8V	12mA(default)
out_port_from_the_led_pio[3]	Output	PIN_AE8		8B8_N0	1.8V	12mA(default)
out_port_from_the_led_pio[2]	Output	PIN_W12		8B8_N0	1.8V	12mA(default)
out_port_from_the_led_pio[1]	Output	PIN_W11		8B8_N0	1.8V	12mA(default)
out_port_from_the_led_pio[0]	Output	PIN_AC10		8B8_N0	1.8V	12mA(default)

图 6-1　表明电路板设计软件与 FPGA 设计软件接口的 . csv 文件

在过去，大多数 FPGA 设计没有信号完整性设计工程师的参与。事实上，现在许多 FPGA 设计仍然没有信号完整性设计工程师的参与。当与 FPGA 器件接口时，电路板设计师往往采用保守的方法进行布局，并假定这种方式可以达到设计要求，这在大多数情况下是适用的。然而，基于本章前面提到的理由，这种方法已经不再适合系统设计的要求了。随着诸如 DDR 2/3 SRAM 这些类型的存储器接口的 I/O 速度的增加，还有高速收发模块，都要求与电路板正确地端接，以防止出现 SI 和 SSN 问题。

按照 FPGA 厂商提供的应用操作说明书中所提供的准则，可以顺利地设计出这些类型的接口。然而，由于每个 PCB 的设计有差异，建议 SI 设计工程师对具有高性能要求的输入/输出进行仿真。这要求 SI 设计工程师、FPGA 设计师及 PCB 设计师之间要进行充分的沟通。SI 设计工程师需要 FPGA 设计师提供 HSPICE 模型和 IBIS 模型，电路板设计师需要提供电路板布局布线等具体细节。SI 仿真往往需要很长时间，因而只能对那些在信号完整性上存在高风险的 FPGA 引脚进行仿真。这是使设计的输入/输出达到高性能的保证。

图 6-2 详细地介绍了 FPGA 设计周期中的每个阶段，不同专业学科的工程师应该在整个设计过程中，自始至终地参与每一阶段的工作。在本章中关于创建

FPGA 引脚设计流程的小节，将对图 6-2 有更加详细的解释。

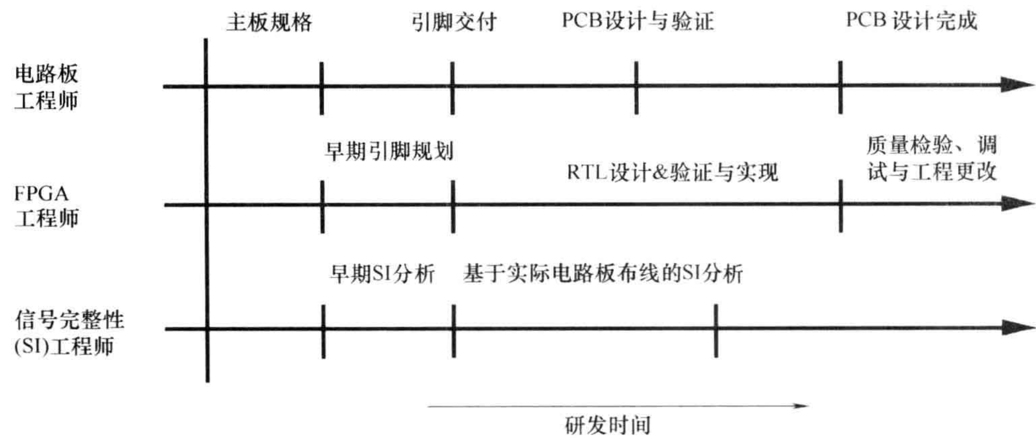


图 6-2　设计周期中不同专业学科参与的详细情况图

6.3　功耗和散热问题

FPGA 的功耗估计有助于指导电路板的电源设计。

6.3.1　滤除电源噪声

为了减少系统噪声，用一个干净而且均匀分布的电源为电路板上所有的器件供电至关重要。在电源线连接到 PCB 节点的附近放置一个 100mF 的电解电容，可以滤除低频电源噪声。若使用稳压器，则应在最后为器件提供电源的 Vcc 和地之间跨接电容。

为了减少电源层的高频噪声，去耦电容应该跨接在电源 Vcc 和地线之间，并尽可能地靠近器件。

6.3.2　电源分配

在电路板上进行电源分配可使用电源总线网络或电源层。虽然电源总线网络是最廉价的解决方案，但它会引起电源压降，所以这种方案只适合用于低成本的双层电路板，如成本比较宽松就应该考虑使用电源层方案。

建议使用两层或更多电源层的方案。电源层应该覆盖电路板整个区域，可以将电源电压 Vcc 均衡地分配给安装在不同位置上的所有器件，这个方案可提供良好的噪声防护。建议把模拟电源层和数字电源层分开，不要共用一个电源层。现实中所有的 FPGA 器件都包含锁相环（PLL）模块，因此在电路板的设计中必须为 FPGA 器件分别提供模拟电源层和数字电源层。

总而言之，对电源分配的建议如下：

—用两个独立的电源层，把数字电源层和模拟电源层分开；
—在锁相环（PPL）的电源层旁放置一个接地层；
—对锁相环（PPL）的电源进行布线时，避免使用多个信号层；
—将数字器件和模拟器件连接到它们各自的接地层上；
—将锁相环（PPL）的（模拟）电源和数字电源隔离。

6.4　信号的完整性

传统的数字设计通常不需要考虑传输线效应产生的影响。但随着系统速度的提高，信号频率变得越来越高，从而对系统造成影响。这意味着设计者不仅要考虑系统的数字属性，还必须考虑高频信号对系统产生的模拟效应。随着I/O接口和存储器接口数据率的增加，特别是嵌入在FPGA器件中的高速收发技术的应用，可能使得高频信号的模拟效应变得尤为突出。传输线效应可能对正在发送的数据产生显著的影响。然而，随着传输速率的提高，高频效应占据了主导地位，即便最短的电路也会遇到例如振铃、串扰、反射和接地反弹这些问题，从而严重地影响了信号的完整性。信号的完整性变差就会导致系统的可靠性变差，造成系统性能降低，最糟的情况是导致系统失效。好消息是这些问题可以通过下面介绍的良好设计技术和简单的布局准则来解决。

6.4.1　信号完整性问题的类型

信号完整性问题通常有四种类型，它们分别是：单个线网的信号完整性，相邻线网间的串扰，轨线崩溃和电磁干扰（EMI）。

6.4.1.1　单个线网的信号完整性

驱动强度是源/漏驱动电流大小的指标，压摆率是源/漏电流变化快慢的指标。这两个指标共同决定了输出信号的上升和下降时间。工艺技术的尺寸越小，允许时钟频率越高，而时钟频率越高意味着信号的上升和下降时间越短。由于制造工艺技术决定了信号的上升和下降时间，这就意味着即使是低频信号，其开关切换时间也缩短了。信号的开关切换时间变短，加上瞬时电流变大，这就造成了更大的开关噪声。对高频链路信号而言，或许有必要缩短信号的上升和下降时间。然而对于低频链路信号，设计师可能要通过延长信号的上升和下降时间来降低噪声。

6.4.1.2　串扰

当信号沿导线传输时，就会在导线周围形成一个磁场。如果两个导线彼此相邻，这两个磁场有可能相互作用，从而导致两个信号能量的交叉耦合，这种现象称为串扰。

以下五种 PCB 设计技术能显著地降低串扰：

1. 在布线约束允许的情况下，信号的线间距应尽量地放宽。

2. 设计传输线时，应该将传输线尽可能地安排在与地线层相邻的层。传输线和地线层的紧密耦合能有效地解除传输线和相邻信号间的耦合。

3. 尽可能地使用差分布线技术，特别是对那些关键的线网。

4. 若信号间有明显的耦合，则令这些信号分布在不同的层并彼此正交。

5. 使得信号之间并行布线的长度尽可能地短。使用短并行线路布线，尽可能地将长耦合的线路最短化。

6.4.1.3 轨线的崩溃（Rail Collapse）

轨线崩溃是指分布在电路板上为芯片供电的电源和地线网络所产生的噪声。输入/输出（I/O）的开关切换会在电源和地线之间的阻抗上产生一个电压降。在实际电路中，这个电压降有可能造成 FPGA 器件供电电压的显著降低，从而使轨线问题进一步恶化。

解决轨线问题的方法是设计更合理的电源和地线分布网络，尽可能地减小电源分布系统的阻抗。

6.4.2 电磁干扰

电磁传导或者辐射会影响电路的正常运行，造成的干扰就是电磁干扰（E-lectro Magnetic Interference，EMI）。这种干扰可能中断、阻止、降低或限制电路的有效性。EMI 源自电流的迅速变化。

FPGA 成为 EMI 源的现象是很罕见的。但随着散热器、电路板和电缆使用的增加，FPGA 产生 EMI 的可能性也增加了。

通过以下措施能降低 FPGA 产生 EMI 的可能性：

1. 尽可能地在接近 FPGA 的位置，用旁路或者“退偶”电容跨接电源的两极；

2. 使用串联电阻控制高速信号的上升时间；

3. VCC 滤波；

4. 屏蔽。屏蔽元器件会增加额外的费用，因此通常作为不得已的最后措施。

电路板 EMI 的两种最常见来源是：

1. 差分信号到共模信号的转换过程中所产生的共模信号会泄漏，并叠加到外部双绞线上。

2. 电路板上的地线反弹会在外部单端屏蔽电缆上产生共模电流 。

将高速信号编组，把它们的布线安排在电路板上不至于产生过多电磁干扰的合适地方，以控制 EMI 效应。

对高速产品进行高效设计的关键是充分利用分析工具的优势，进行准确的性

能预测。测量 EMI 的大小则是验证设计过程、降低风险以及增加对设计工具信心的一种手段。

6.5 FPGA 引脚分配的设计流程

这里为在电路板设计中顺利分配 FPGA 引脚推荐两个设计流程。这两个设计流程都能为 FPGA 设计师和电路板设计师间的交流提供充分的沟通机制。

6.5.1 流程1：由 FPGA 设计师主动

在这个设计流程中，最初的 FPGA 引脚分配方案是由 FPGA 工程师制定的，他将 FPGA 引脚分配的细节提供给 PCB 设计工程师。为方便电路板设计，电路板设计工程师若想改动引脚的分布，就要将这些改动的细节提供给 FPGA 设计师。之后，FPGA 设计师再用 FPGA 设计软件对引脚的分配做修改，并确认这些改动不会影响 FPGA 设计的正常运行。这个过程一直持续下去，直到得到最终的引脚分配方案，既能满足 FPGA 设计师的需求，也能达到 PCB 设计师的要求。

在实际工作中，如果最初的引脚分配方案是由 FPGA 设计师提出的，那么他首先需要了解电路板的布局，譬如与 FPGA 接口的电路板器件（如存储器，收发器，微处理器等）有几个，它们的位置在哪里。然后，他才能根据实际情况进行引脚分配。譬如，先将存储器接口分配给某个 I/O 区，接着让 FPGA 设计软件自动地进行引脚分配。这种方法可以大大加快引脚的分配过程。这是因为 PCB 设计师为了减少电路板上的线路交叉等事项，只需要与 FPGA 设计师，就少量引脚的交换做一些简单的沟通即可，不需要大规模地修改引脚的分配方案（见图6-3）。

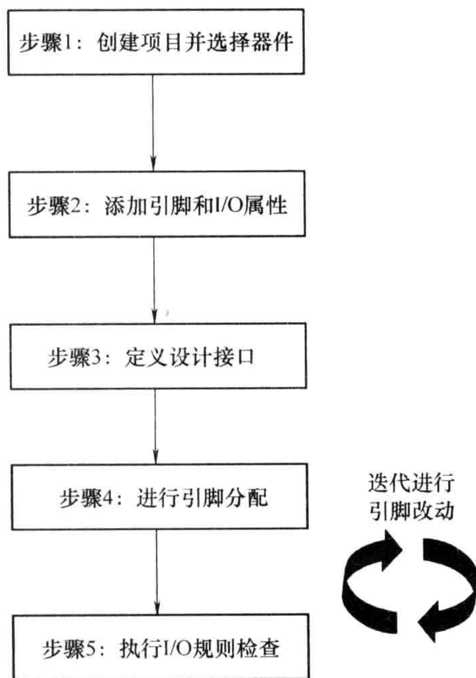


图6-3 由 FPGA 设计师主动的 FPGA 引脚分配流程图

步骤1：第一步是在 FPGA 设计软件中进行的。FPGA 设计师首先选取一款型号和封装合适的 FPGA 器件，在这个目标器件上创建一个 FPGA 设计项目。为适应未来设计的扩展或缩减，建议设计者在这个阶段把 FPGA 设计

软件中允许移植到所有器件上的功能选项启动。

步骤 2：根据 FPGA 设计，FPGA 设计师开始输入引脚信息。在 FPGA 设计周期中的这个阶段，FPGA 设计未必完整，但接口却必须先确定下来。至少顶层设计文件应当存在。这个文件为设计师提供充足的信息，以输入引脚名称，输入诸如 I/O 标准、电流强度等引脚属性。这些信息可手动输入到 FPGA 设计软件中，在大多数情况下也可从其他资源中导入，例如 Microsoft Excel。建议在设计说明书中定义这些信息并使其具有 . csv 格式，以方便导入 FPGA 设计软件。这将大大缩短引脚信息的输入过程，并降低人为错误的风险。

如果正在使用接口 IP，其中一些 IP 可能已经含有引脚属性信息。那么在设计中应当添加这些 IP 的源文件。FPGA 设计软件通常能从 IP 源文件中读取引脚的属性。

步骤 3：通过配置每个被使用 IP 的端口和参数来定义设计接口，使 IP 的端口与顶层 HDL 文件的接口连接。正如前面提到的那样，建议顶层设计文件早已确定。然而在设计说明书已经完成，但设计文件并不存在的情况下，有些 FPGA 设计软件也可以根据输入 FPGA 设计软件的引脚信息，自动地生成顶层 HDL 包装文件。在 FPGA 设计软件中，需要启动对顶层设计文件的 I/O 规则检查。设计师通过编写设计接口，实际上就为 FPGA 设计生成了顶层模块的接口方框图。给 FPGA 设计软件提供的设计信息越多，FPGA 设计软件可执行的 I/O 规则检查就越完整。

步骤 4：进行引脚分配。如果分配给某 I/O 信号引脚的确切位置是已知的，就直接将已知的信息输入到 FPGA 设计软件中。对于 IP 来说，这些信息通常是被导入的。如果只知道信号引脚的大致范围，那么分配给该信号的引脚位置可以不必十分具体，只要指定一个范围即可，例如把某信号分配到 I/O 区域 1，随后，让 FPGA 设计软件自动地选择引脚的具体位置。

步骤 5：进行输入/输出（I/O）规则检查，产生合理有效的引脚分配。所有 FPGA 设计软件都有 I/O 规则检查功能。这一功能可用来检查 I/O 信号引脚分配的合理性和有效性。有些 FPGA 软件设计包能根据器件引脚的分区自动为 I/O 信号分配合适的引脚，而不必逐一指定具体的引脚。自动分配产生的引脚列表可以在被用户接受后，传递给电路板设计师，用作与电路板连接的引脚分配表。

在设计尚未完成的情况下，用设计软件对 FPGA 引脚分配做 I/O 规则检查时，可选的检查项十分有限。因此，强烈建议设计师先编写一个虚拟设计，其中包含接口所使用的每个 IP 和时钟网络的细节。这些接口能与虚拟逻辑连接，譬如，FIFO 就是这样一个接口，与 FIFO 连接的内部模块尚未完工，可用虚拟模块代替。借助于这种方法，FPGA 设计软件可以对设计中将用到的所有 I/O 规则做全面的检查，从而确保在以后的设计中添加内部逻辑块后，仍然能使用相同的引

脚分配。

重复执行步骤 4 和步骤 5，直到 FPGA 和电路板都可用的 FPGA 引脚分配方案瓜熟蒂落。

随着设计的逐渐完善，任何潜在的引脚分配问题都应及时与电路板设计师沟通。可以在电路板层面，也可以在 FPGA 设计层面上做修改来解决这些问题。但不需要对虚拟设计做任何改动，因为虚拟设计只是用于表示最终设计将如何分配 FPGA 的引脚。

6.5.2　流程 2：由电路板设计师主动

在这个设计流程中，由 PCB 设计师用电路板设计软件生成最初的 FPGA 引脚分配方案，并把其细节提供给 FPGA 设计师。电路板设计师也可运行 FPGA 设计软件来输入引脚的详细信息。但这种情况在现实中很少见，除非 FPGA 和电路板都由同一工程师负责设计。在大多数情况下，由 FPGA 设计师用 FPGA 设计软件进行引脚分配，并确认该引脚分配方案能满足 FPGA 设计的需求。若引脚分配存在问题，则 FPGA 设计师应先用 FPGA 设计软件进行修改，引脚分配正确之后，再将引脚变动的信息传送给电路板设计师。这个过程一直持续下去，直到获得一个既能满足 FPGA 设计师又能满足电路板设计师需求的最终引脚分配方案（见图 6-4）。

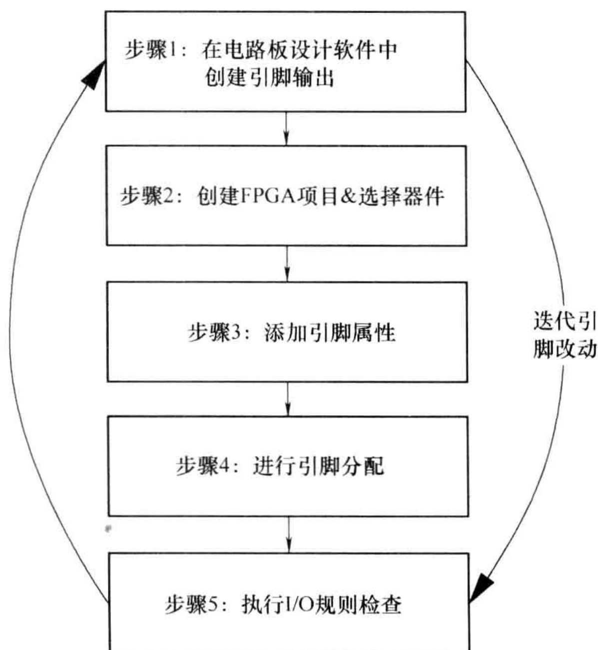


图 6-4　由电路板设计师主动的 FPGA 引脚分配流程图

步骤 1：根据电路板上将与 FPGA 接口的元件，由电路板设计师，编写 FPGA 的引脚分配方案。这需要关于 FPGA 器件驱动能力和时钟限制的详细信息。在实际工作中，为了走好这一步，电路板设计师需要与 FPGA 设计师密切配合，咨询收发器在 FPGA 器件上的具体位置，电源轨线的具体要求，以及与引脚输出限制有关的其他问题。然后，由电路板设计师编写第一版引脚分配方案，并将引脚分配信息传给 FPGA 设计师。

步骤2，3，4：和流程1中的步骤1，2和4相同。FPGA设计师将用FPGA设计软件创建FPGA项目，进行引脚分配，并配置引脚属性。

步骤5：由FPGA设计师运行I/O规则检查器以确认引脚的分配，提出改进的建议，并与电路板设计师沟通。这个过程一直持续下去，直到获得到一个满意的引脚分配为止。像流程1一样，FPGA设计师应当创建一个虚拟设计或使用真实的设计以确保该引脚分配方案能正常运行。

6.5.3 FPGA 设计师和电路板设计师如何进行引脚改动的沟通

通过口头或者电子邮件就引脚分配方案的变动进行沟通是一种趋势。然而，这种沟通方式很容易出错。需要用一个有版本控制的正式文件，在FPGA设计师和电路板设计师之间，作为传递引脚变动信息的媒介。在本章前面曾提到，许多公司中常用 Microsoft Excel 来满足这个需求。使用 Microsoft Excel 的一个优势是很多电路板设计工具和一些 FPGA 设计软件都能导入或导出 .csv 文件。

6.6 电路板设计的审查要点

为了保证FPGA引脚分配获得成功，电路板设计审查要点总结如下：

1. 对电路板设计进行电源热分析，以确保所有电源层在提供所需最大电流时，仍能保证电压轨线符合规范。

2. 进行引脚分配检查。

   a. 用FPGA设计软件检查引脚的分配；

   b. 将所有未使用的输入端接地；

   c. 根据需要将所有未使用的输入/输出（I/O）接地、接 Vcc 或挂起；

   d. 检查每个I/O区VCCIO的正确性；

   e. 设计是否符合SSN准则？

   f. 选择可移植的器件以适应未来设计规模的扩大或缩减。

3. 对照厂商提供的配置手册，进行配置模式检查。

4. 对照厂商推荐的电源设计标准，检查电源的连接和去耦。

5. 进行电路板信号完整性仿真。

6. 将I/O时序和I/O时序需求进行对比。在进行对比时，要求整个设计已接近完成或至少设计的I/O接口部分已经完成。

7. 将电路板设计递交给FPGA设计小组和PCB设计小组，共同完成对电路板设计的全面审查。

第 7 章　功耗和热分析

7.1　引言

FPGA 器件规模的增长和性能的提升导致 FPGA 器件功耗变大。FPGA 设计工程师和 PCB 设计工程师在选择使用 FPGA 器件和特定的 FPGA 供应商时，都需要考虑功耗问题。因为 FPGA 器件的功耗将影响 PCB 的电源设计，以及对稳压器、吸热设备和系统冷却系统的选择。简而言之，对整个系统做功耗预算至关重要。

对功耗敏感和功耗预算较紧的应用而言，在设计开发的过程中，设计工程师需要进行功耗分析，并采用适当的低功耗技术。在整个设计周期中，工程师应具有细化预估功耗的能力，并采用恰当的功耗管理设计技术。

如今的 FPGA 器件拥有多种特性来降低 FPGA 器件的功耗，包括 FPGA 设计软件中的功耗优化选项。本书中关于 RTL 编码指南和时序收敛的章节将涉及功耗优化技术的细节。

FPGA 厂商也在设计流程的不同阶段提供了 FPGA 功耗估计的解决方法。

本章将首先考察影响 FPGA 器件功耗的几个基本要素，以及影响设计者获得设计功耗准确估计能力的主要因素；接着，关注在设计最初阶段进行功耗估计时所使用的工具和软件，设计最初阶段进行功耗估计是为了正确地选择 FPGA 的设计技术，以及为电路板设计选择恰当的电源稳压器和元器件；然后考察可对设计实现进行更加精细功耗估计的工具和技术。最后，对于 FPGA 设计中的不同阶段处理功耗问题的具体方法进行总结，以表格的形式提出了最佳的建议。

7.2　功耗的基本要素

热功耗（Thermal power）是总功耗中耗散在器件封装上的那一部分功耗。为保证管芯结区内部温度维持在推荐的范围内，在决定是否应该对 FPGA 采用散热措施（如散热设备）时，需要考虑热功耗。

当考虑 FPGA 器件的输出负载和外部的终端连接时，FPGA 器件的总功耗由以下五个主要的部分构成。

7.2.1　静态功耗

所谓的静态就是设计中没有活动和翻转时的状态。当设计处于静态时由漏电流引起的功耗称为静态功耗。这种类型的功耗通常被称为待机功耗，它与具体的设计无关。漏电流的大小取决于管芯的尺寸、结区的温度和工艺的变化。这些资料可以从 FPGA 器件的数据手册或者从厂商提供的早期功耗估计试算表中提取。这里推荐后者，因为后者的数据格式通常比大多数数据手册清晰得多。

7.2.2　动态功耗

由于 FPGA 内部节点翻转引起器件操作所消耗的功耗称为动态功耗。也就是，逻辑阵列中负载电容的充放电和信号路由引起的功耗。影响动态功耗的主要变量是充电电容，供电电压和时钟频率。FPGA 器件的总动态功耗中很大部分源于它的布线结构。

动态功耗取决设计，RTL 代码的编写风格对动态功耗有很大的影响。

7.2.3　输入/输出功耗

连接到器件输出引脚的外部负载电容的充放电和所有端接网络消耗的功率就是输入/输出（即 I/O）功耗。同样，I/O 功耗取决于设计，并受到 I/O 标准、数据速率、引脚配置情况（输入，输出，双向）的影响。输入的端接和输出的电流强度、压摆率和负载都会影响 I/O 功耗。

7.2.4　浪涌电流

浪涌电流是器件在初始上电时需要的电流，因而会产生功耗。在上电阶段的一段特定的持续时间内，必须提供给器件最小的逻辑阵列电流（ICCINT）。这段持续时间的长短取决于可以从电源获得的电流的大小。当电压达到额定电压的 90% 时，通常就不再需要这个初始的强电流了。随着器件温度的升高，上电时需要的浪涌电流会减小，然而待机电流会变大。

7.2.5　配置功耗

在配置器件时需要的功耗就是配置功耗。在配置和初始化阶段，器件需要功耗来复位寄存器、使能 I/O 引脚、输入操作模式。在器件上电阶段，配置器件之前和期间，I/O 引脚通常处于三态以减少功耗并防止它们在此期间失控。

7.3 准确估计功耗的关键因素

在讨论对 FPGA 设计进行功耗和热分析的最佳方法之前，我们先来关注影响准确估计功耗的几个关键因素（见图 7-1）。

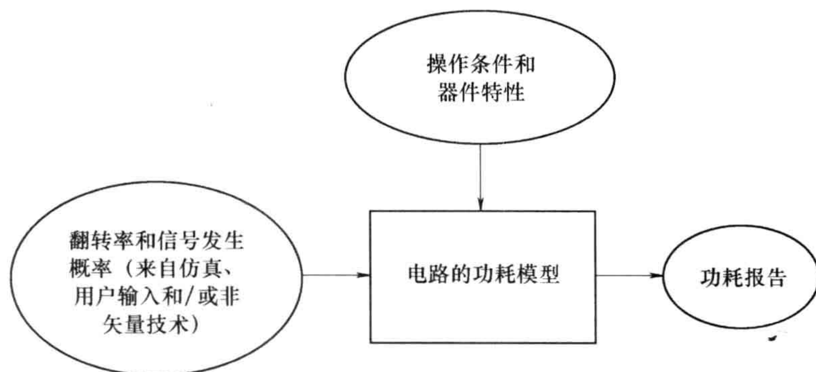


图 7-1　准确估计功耗的关键因素

7.3.1 FPGA 电路的准确功耗模型

功耗估计模型是 FPGA 厂商提供的功耗估计解决方案中的一部分。FPGA 设计者必须相信 FPGA 厂商提供的模型是可靠的。这些模型通常基于 HSPICE 开发并和芯片的特性相关。不同 FPGA 厂商的开发过程略有不同。功耗模型的准确度随 FPGA 器件系列的不断成熟而变化。新上市的 FPGA 器件系列提供的是初步的功耗模型，功耗模型会随 FPGA 厂商对这个器件系列特性的完善而改变。若 FPGA 厂商在研发初期提供的 HSPICE 模型是保守的，则功耗模型的改变带来的负面影响就非常小。向器件厂商咨询他们如何研发功耗模型的细节，有助于对功耗模型的准确度建立合理的期望。

7.3.2 每个信号的准确数据切换率

数据切换率也被称作信号活动性，与设计的性能相关。时钟速率固然重要，但是单位时间内信号变化的平均次数更加重要，因为这个切换过程会影响功耗。

逻辑“1”消耗的功率超过逻辑“0”消耗的功率，因而信号维持逻辑“1”时间的长短将影响功耗。这就对使用端接标准的输入/输出功耗有影响。

由于数据切换率取决于系统的运行，因而它完全在 FPGA 设计工程师的掌控之中。数据切换率信息通常从设计仿真中抽取或者根据以往的设计经验。因此对由以前完成的工作继承而来的设计可以直接输入一个合理的数据切换率，与新设计比较是件较容易的事。但也不能过分强调使用数据切换率在反映最终系统运行

活跃性中的重要性，毕竟对数据切换率的预测不太准确是功耗估计误差的主要来源。

在很多情况下，仿真数据不能代表实际的操作。如执行的仿真是以测量代码覆盖率为目的，就容易造成过高估计实际操作中的功耗。作为设计者应当避免由于过低预测数据切换率造成的危险，因为这样将导致功耗的低估。然而，过高预测功耗可能导致更加昂贵的功耗管理方案。

除非 FPGA 设计工程师设定，在 FPGA 厂商提供的功耗估计方案中，假设默认的数据切换率为 12.5%。对于大多数应用而言，这样的假设在设计周期的起始阶段已经足够。由于大多数设计中并不是在所有的节点上的数据都有很高的切换率，所以在最终的应用场合，规定将总功耗的 30% 以内作为功耗估计误差的余度，应该能满足一般需求。然而，这一规定并不适用于那些设计主体进行高性能处理的情况，例如许多 DSP 处理的应用场合。这些设计通常呈现出较高的数据切换率。

在 FPGA 厂商提供的功耗估算工具中修改数据切换率非常容易，可以快速地看到数据切换率对功耗的影响。建议根据不同的应用场合，设计者应尽量准确地估计数据切换率。如果确实不清楚数据切换率的大小，建议试用一个数据切换率的范围来估算可能的最好和最差的功耗情况。注意：完整的系统设计其数据切换率不可能高于 40%。

7.3.3 准确的运行条件

从图 7-2 中可以看出温度对待机功耗的影响，当结温高于 85℃时功耗会有急剧地增加，特别是对于那些 65nm 工艺尺寸以下的器件。

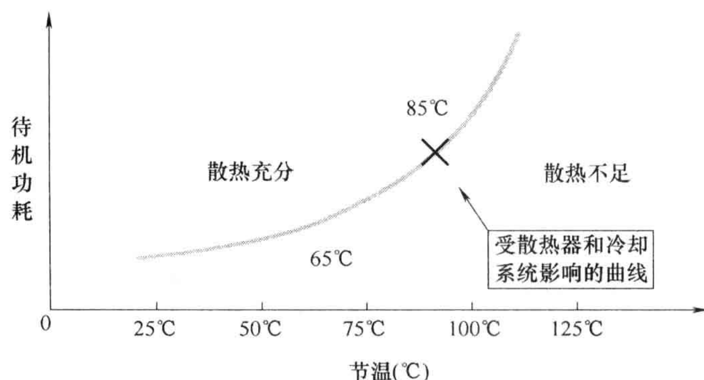


图 7-2　待机功耗和温度

静态功耗受温度的影响很大，泄漏功耗是结温的指数函数。当泄漏功耗增高时，结温也随着增高，而温度增高又会进一步增大泄漏功耗，形成了正反馈。$T_j = T_a + \theta_{ja} \times$（待机功耗+动态功耗），$T_a$ 代表器件外界环境温度，θ_{ja}是器件连接点和周围空气的热阻。确保结温维持在其操作允许的温度范围内，而不至于使其进入正反馈至关重要。器件消耗的功耗越大，产生的热量就越多，若想要维持正常的操作温度，则必须将这些热量散发掉。

对 FPGA 设计者和电路板设计者来说，使用合适的热管理技术来降低功耗非常重要。

7.3.4　资源利用

FPGA 器件的资源利用率是影响功耗的第四个要素。通常使用的逻辑单元越多功耗就越大。

当然，作为设计者应当知道 FPGA 器件中不同类型的资源所消耗的功耗有所不同。作为项目执行者应当有能力选用合适的资源以减少功耗，例如在决定究竟选用逻辑单元还是专用硬件模块（如 RAM 和 DSP 模块）来实现相同功能时，应做适当的权衡。

典型的 FPGA 设计中约 65% 的功耗是内核的动态功耗，约 24% 是内核的静态功耗，约 10.5% 是输入/输出动态功耗，约 0.5% 是输入/输出静态功耗。

如果更加仔细地探究内核的动态功耗，可以发现其主要部分源于逻辑单元中的信号路由和组合逻辑。RAM 模块也消耗可观的动态功耗。

时钟网络的动态功耗包含全局时钟的布线资源加上分布在 LEs、RAM 和 DSP 模块中的本地时钟布线资源所消耗的功率。设计者可以通过选择资源类型和使用时钟控制模块来控制动态功耗。这部分内容将在第 12 章中详述。

7.4　设计周期早期的功耗估计（电源规划）

如前面所述，FPGA 厂商提供的数据手册中关于其系列器件典型功耗方面的资料不太全面。然而，FPGA 厂商确实为它的某些器件提供了可产生功耗报告的功耗估计工具。

对 FPGA 的功耗提前作出估计有助于指导电路板的电源设计。功耗估计通常需要在 FPGA 设计完成前，甚至设计刚开始时就着手执行。FPGA 厂商提供的功耗估算表可在设计周期中的不同阶段用来估算用户设计的功耗，并对其进行初步的热分析。

图 7-3 展示了一个功耗估算示例，通过人机对话填写表格，便可自动对 Altera Stratix IV GX 系列进行功耗估算。

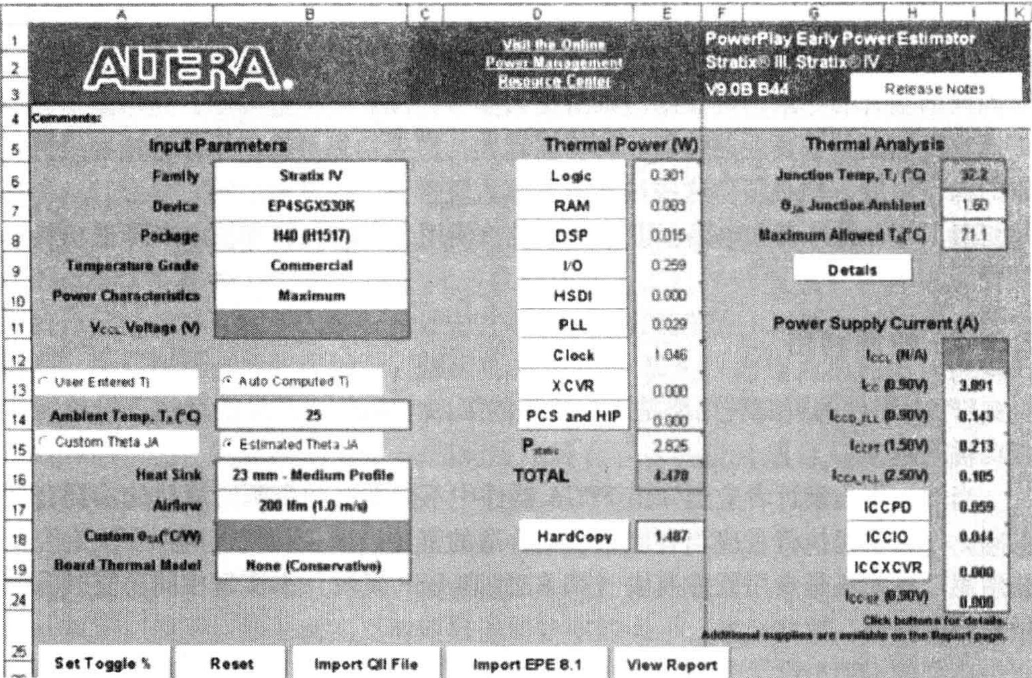


图 7-3　Altera Stratix IV GX 系列功耗估算表示例

FPGA 厂商提供基于 Excel 的功耗估算表，用户可从该厂商的官网上免费下载。操作环境和最终设计的相关信息提供得越多，功耗估计就会越准确。器件的成熟度也会影响功耗估计的准确性，例如厂商提供的功耗模型是最终的还是初步的，功耗模型不同，功耗估计的准确性就不同。通过功耗估算表，花最少的功夫就可以得到大致正确的功耗估计，通常可以将误差控制在实际功耗的 30% 以内；这样用户就可为自己的项目选择一款合适的 FPGA 器件，并提前确定电源的规格。多花一些时间、多输入一些关于设计和操作条件的详细信息，通常能获得误差在实际功耗 20% 以内的估计。设计者可在这些工具中输入设计和操作条件的详细信息。一些 FPGA 厂商提供的工具能将设计编译中生成的数据自动导入到功耗估算表中。在局部设计的功耗估计中以及基于传承信息的功耗估计中，这个功能表现得十分良好。将这些信息作为一个起点，而将诸如不同资源的数量，时钟个数等细节在功耗估算表中进行编辑，以反映最终设计预计的大小和特性。正如 7.5 节中讨论的那样，使用这种由 FPGA 厂商提供的功耗估计和分析方法输入设计数据，是一种快得多并且更不易出错的方法。

7.5　基于仿真的功耗估计（设计的功耗验证）

假如仿真向量代表了实际系统的运行情况，那么这种基于仿真的功耗估计方法将是最准确的功耗估计解决方案。基于仿真的功耗估计利用由标准 EDA 工具，如 Mentor Modelsim，Synopsys VCS 和 Cadence Incisive 等产生的仿真结果来模拟 FPGA 器件的运行。可把仿真得到结果当作激励，输入到由 FPGA 厂商提供的基于仿真的功耗估计工具中。

将数据从 EDA 仿真工具转到 FPGA 厂商提供的软件，通常使用向量变化存储（Vector Change Dump，VCD）文件。FPGA 厂商提供的功耗估计软件中的功耗估计解决方案比功耗估算表方法更加准确，原因是在设计的布局和布线都全部完成之后，FPGA 软件的功耗估计模型中考虑了设计中实际使用的布局布线模型。此外，使用的测试向量与真实运行情况的符合程度对功耗估计的准确度也有很大的影响。

拥有一个有准确仿真向量的设计意味着设计已经完成或者马上就要结束。因此为了确定设计的真实功耗，建议大部分设计在接近设计后期进行此类分析。相比在整个设计周期中不断地检查设计是否在功耗预算之内，这样做更加明智。

功耗临界（敏感）的设计是一个例外，由功耗估计得到的数据可用来决定是否需要对 RTL 代码进行优化以降低功耗，是否需要使用 FPGA 厂商所提供软件中的功耗优化选项对 RTL 代码进行优化以降低功耗。在设计早期对已有的 RTL 模块可运行基于功耗估计的仿真，以期得到这些模块的翻转率，翻转率用在功耗估算表中对已有的 RTL 模块进行功耗估计。可重用 IP 模块的功耗报告也要纳入模块的文档中，给使用这些设计模块或 IP 的其他用户提供有关模块期望功耗的背景信息。

通过仿真可以得到设计中每个节点的数据切换率，所以最准确的功耗估计可以通过门级仿真得到。然而，想通过门级仿真得到最准确的功耗估计十分困难。因为对于如视频和图像处理等应用场合而言，门级仿真的运行时间特别冗长，尽管这种分析方法可得到最准确的功耗估计结果，但因为仿真时间太长，通过门级仿真来估计功耗并不实用。因此，对这类应用场合，我们建议采用 RTL 级仿真。而门级仿真仅用于个别的运行条件，对设计做一些合理性检查。当然，对最终的应用场合而言，如果仿真时间的长度能被接受，建议使用门级仿真。

通过 RTL 仿真，设计者可以得到被仿真系统中输入/输出引脚和大多数寄存器的数据翻转率，而且相当正确。而综合器在综合优化的过程中，会进行寄存器的复制和合并，因而寄存器的个数可能会有一定程度的误差。由于综合过程中执行了优化操作，造成节点名称的不匹配，使得组合节点的个数也不十分准确。然

而这并不是什么大问题，因为大多数基于仿真的功耗估计工具都包含一个称为无向量估计的模式。该模式与基于 RTL 仿真的功耗估计方法结合在一起，可以为设计者提供准确度可接受的功耗估计值。

无向量功耗估计使用统计分析的方法来预测节点在已知完好的数据点间的翻转概率。从图 7-4 可以看到，如果知道输入 A，B，C，D，E，F，G 和 H 的静态概率和翻转率，就有可能估计出 I，J，K，L 的静态概率和翻转率；同理，也可以估计出最终输出 M 的静态概率和翻转率。

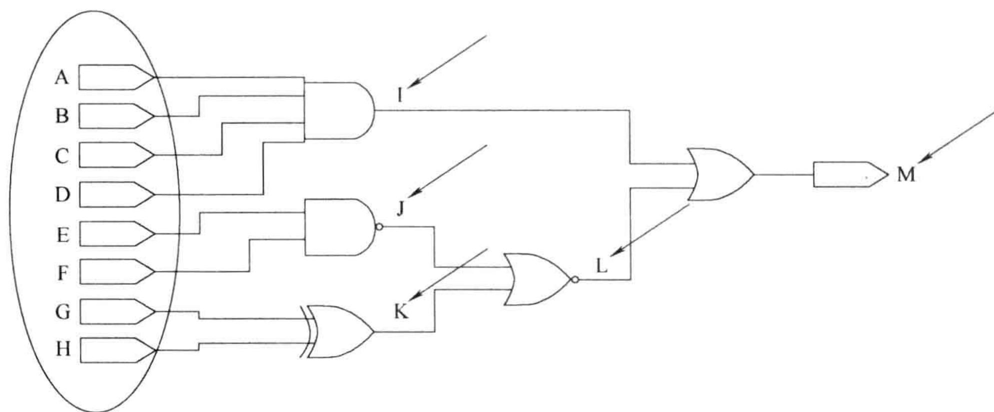


图 7-4　节点翻转概率

这个方法可以用来提高基于 RTL 仿真的功耗估计方法的准确度。为了达到最佳实际效果，建议运行门级仿真的例子，但是对于仿真时间很长的系统，建议使用 RTL + 无向量估计的方法。在整个设计过程中某些检查点上，也建议使用基于门级仿真的功耗估计。实际上，项目已进展到这一阶段，对其做基于门级仿真的功耗估计更多的应该是合理性验证，而非必不可少。在进行早期的功耗估计之后，应当在功耗预算上留足够的余量，这样就不必经常地对设计进行功耗优化。在早期功耗估计中，设计者需要改变温度和电压等操作条件，以确保估计得到的功耗值确实能反映真实环境下的功耗。

基于仿真的功耗估计工具产生的功耗报告，可以帮助设计者确定系统在散热和电源规划方面的需求。这些报告详细地指出哪些器件结构，甚至哪个层次的模块的热能耗最大，以便于做出降低功耗的正确决定。在数据切换率相当准确的前提下，这种功耗估计方法通常能得到误差在 20%（器件测量值）以内的高质量功耗估计值（见图 7-5）。

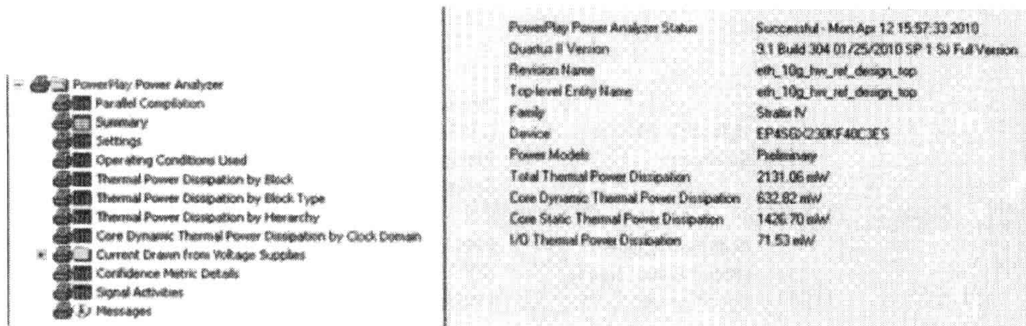


图 7-5　Quartus II PowerPlay Estimator 功耗估计工具的报告示例

7.5.1　局部仿真

基于仿真的功耗估算工具所面临的困难之一，就是需要剔除测试平台中的仿真初始化时间。如果测试平台中的仿真时间并不只是系统长时间的持续操作，而是包括了仿真初始化所需要的时间，那么系统有效的数据切换率就会降低。如所进行的仿真并没有将整个仿真时间都用于信号活动性的计算，就会降低功耗估算的准确度。例如，仿真时间为 10000 个时钟周期，前 2000 个时钟周期都用在芯片的复位上。如果信号活动性计算使用所有的 10000 个时钟周期，那么数据切换率只有它们稳态值的 80%（因为前 20% 的仿真时间用在芯片复位上）。一些 FPGA 厂商提供的功耗估计工具，允许用户只把指定的 VCD 文件中的有用部分用于功耗分析，这就可以把仿真初始化阶段所花费的那部分时间剔除出。

7.6　功耗估计的最佳实践方法

功耗估计的最佳实践方法见图 7-6。

设计周期阶段 (Stage of Design Cycle)	任务 (Task)	工具 (Tools)	附加内容 (Additional Conttnt)
			历史设计 (Legacy Designs) 以往经验 (Previous Experience) 设计说明书 (Design Specification)
	选择器件 (Device Selection)	FPGA厂商功耗估算表 (FPGA Vendor Power Estimation Spreadsheet)	早期的RTL代码 (Early RTL code)
	电路板供电说明书 (Board Power Supply Specification)	FPGA电路板设计指南 (FPGA Vendor Board Design Guidelines)	
		FPGA厂商功耗估算表 (FPGA Vendor Power Estimation Spreadss heet)	
早期功耗估计阶段 (Early Power Estimation)	电路板设计 (Board Design)	FPGA电路板设计指南 (FPGA Vendor Board Design Guidelines)	
	基于开展设计进行功耗抽查 (Spot check Power Based Upon Evolving Design)	FPGA厂商功耗估算表 (FPGA Vendor Power Estimation Spreadsheet)	硬件描述语言设计 (HDL Design)
设计开展阶段 (Evolving Design)	功耗优化后估计功耗 (Estimate Power for Power Optimization)	FPGA厂商提供的基于仿真的功耗估计工具 (FPGA Vendor Simulation Based Power Estimation Tool)	测试平台 (Testbench) EDA仿真工具 (EDA Simulation Tools)
	确定最终功耗 (Determine Final Power)		硬件描述语言设计 (HDL设计)
	功耗优化后估计功耗 (Estimate Power for Power Optimization)		测试平台 (Testbench)
最终设计阶段 (Final Design)	电路板功耗测试 (Measure Power on Board)	FPGA厂商提供的基于仿真的功耗估计工具(FPGA Vendor Simulation Based Power)	EDA仿真工具 (EDA Simulation Tools)和最终的电路板和测试设备 (Final Board and Test)

图 7-6　功耗估计的最佳实践方法

第 8 章　RTL 代码设计

8.1　介绍

无论是为 FPGA 器件还是为专用集成电路（ASIC）编写 RTL 代码时，设计师都不能不回答以下五个共同的高层次问题：

1. 设计模块想要达到的最终目标究竟是什么？

2. 设计模块究竟想要达到最高性能，还是实现最小面积？

3. 模块代码的功能正确吗？是否能很容易地用目标器件的综合工具进行综合？

4. RTL 模块的代码能不能重复使用？

5. 设计模块能否很容易地完成布局布线，成功地转变为组件模型？

然而，在为 FPGA 编写 RTL 代码时，还需要回答另外两个 FPGA 特有的高层次问题：

1. RTL 代码是否已针对某目标类型的 FPGA 器件结构进行过优化？该代码是否还可以移植到其他结构不同的 FPGA 上？

2. RTL 代码是否已针对编译时间进行过优化？

当更深入地探究如何为 FPGA 编写 RTL 代码时，我们将发现 FPGA 的 RTL 代码与 ASIC 的 RTL 代码相比较，有很多不同之处。这些相异之处源于不同 FPGA 器件与 AISC 之间的结构差异。这就给我们提出了在编写 FPGA 器件的 RTL 代码时的第一条准则，即“理解目标 FPGA 器件的结构”。

本章为具有不同背景的设计师们提供了快速上手的窍门。在全面讲解编写 RTL 代码常用的良好习惯之前，本章描述了一些常见的 FPGA 器件结构。然后给出针对 FPGA 器件结构优化的 RTL 编码规范，最后列出了为 FPGAs 器件编写 RTL 代码的十七条建议作为本章的总结。

8.2　常用术语

硬件描述语言（Hardware Description Language，HDL）：是一门用来对硬件进行建模的软件编程语言。

寄存器传输级（Register Transfer Level，RTL）：根据信号和寄存器值的数据

流操作来定义输入和输出之间的关系。

行为建模（Behavior Modeling）：通过输入输出关系的描述来建立电路模型。行为建模只描述电路的功能，而不描述最后实现的电路结构。它没有在某种硬件上实现的明确意图，而且其编码风格也是一般性的，以致于它能够以任何技术为目标（见图 8-1）。

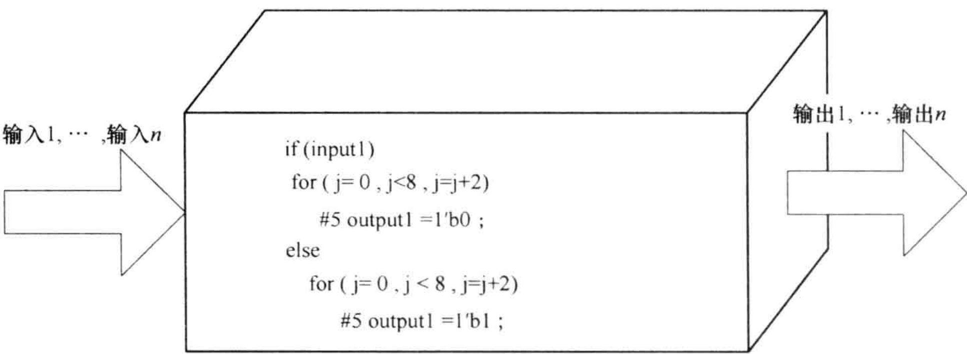


图 8-1　行为建模

结构建模（Structural Modeling）：通过描述较低层次的元件和原语的互相连接关系来建立电路模型。结构建模既描述电路的功能也描述电路的结构。

它由设计师脑海中的硬件实现的想法所创建（见图 8-2）。

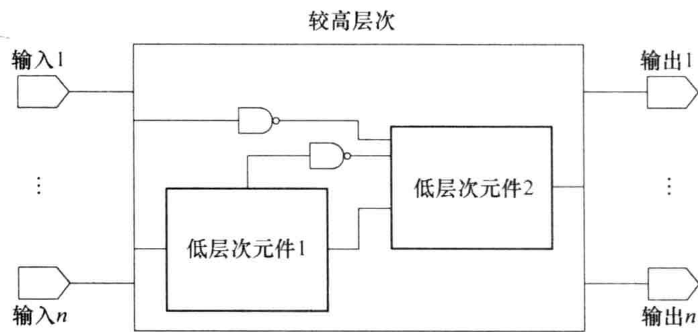


图 8-2　结构模型

综合（Synthesis）：把 HDL 代码转换成电路的处理过程，转换之后还要对电路进行优化。综合的主要任务是解释设计师编写的 RTL 代码，根据 FPGA 目标器件的结构生成相应的硬件。为了生成正确的逻辑，综合工具要求 RTL 代码有特定的编码风格。编码风格对于生成高速有效的逻辑至关重要（见图 8-3）。

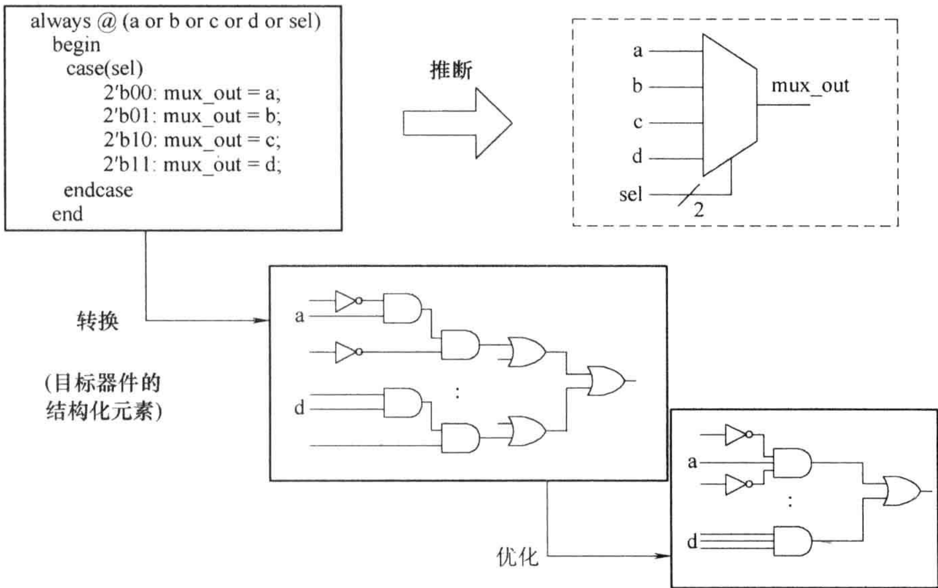


图 8-3　综合

8.3　工程师对有 ASIC 设计背景的建议

对具有 ASIC 设计背景的工程师来说，首先需要知道的事情是 FPGA 器件内部布满了寄存器。无论寄存器是否被使用，它都存在于已购买的 FPGA 器件中。建议设计师把寄存器看作免费资源，所以要尽量使用它们，否则就浪费了。

寄存器的使用对 FPGA 设计的性能而言非常重要。在相同的工艺条件下，FPGA 的逻辑一般要比 ASIC 的逻辑慢。为了满足设计性能的要求，可以利用寄存器使设计流水化。

许多 ASIC 设计中使用锁存器。在 FPGA 设计中不要使用锁存器，而要使用寄存器来代替它。这将显著地提高 FPGA 的时序性能，即便付出潜在的延迟代价。

在 ASIC 设计中，为降低功耗并提高可测试性经常使用门控时钟技术。而在 FPGA 设计中，尽量不要使用门控时钟，而要使用“时钟使能”。低偏斜时钟网络是高性能设计运行的关键，而 FPGA 器件内这样的时钟网络数量有限。使用门控时钟会用完低偏斜全局时钟，从而限制设计的性能。FPGA 器件的所有寄存器均可使用时钟使能信号。在不对设计性能造成不可恢复破坏的前提下，时钟使能信号可用来降低功耗，并提高设计功能的可测试性。

不能将 FPGA 器件中的缓冲器当作可靠的延迟线网用以提高设计的性能。因此，当遇到设计比较临界的部分时序时，最好保守一些，在时序需求方面要留有

足够的余量。

尽管设计师可以为 FPGA 器件中的资源买单，但是只要确定了使用某个 FP-GA 器件来实现设计，那么可用的资源只能局限于目标器件的规模，不管是否使用都得花费那么多钱。设计师可用的逻辑单元、存储块和乘法器数量都将受到目标器件的限制。此外，FPGA 器件中布线资源也是有限的。当设计达到器件利用率的上界时，设计师可能看到设计的性能开始下降。

8.4 推荐的 FPGA 设计规范

8.4.1 同步与异步

总而言之，在设计实践中必须采用同步设计方法，因为它将自始至终地帮助你达到设计目标。

而异步设计技术会使电路的行为依赖于器件中的传输延迟，造成时序分析的不完整，产生冒险竞争。

在同步设计中，只用一个时钟信号来触发所有的事件。只要所有寄存器的时序要求都能够得到满足，那么，无论采用什么工艺、电压和温度（PVT）条件来实现设计，该同步设计的行为都是可预测的、可靠的。这就是说，采用同步设计方法完成的设计可以很容易地移植到系列或速度等级各不相同的目标器件上实现其功能。

8.4.2 全局信号

FPGA 设计软件将自动地选择全局布线资源。由于全局信号资源有限，应当被视为珍贵资源。建议尽可能地限制时钟域的数量。虽然设计师可以选择由自己来控制全局资源的使用，但想要得到比由软件自动处理的结果更好一些，几乎不可能。

设计师必须为 FPGA 设计选择一个同步或者异步的复位方案。FPGA 设计必须有一个可将整个电路置于确定初始状态的系统复位信号。设计师应该在仿真开始时刻，将该系统复位信号输入到仿真测试平台，用以验证其操作的正确性。

如果不能确定哪一种复位方案对系统来说更好一些，就使用同步复位，因为它更容易理解。

如果决定使用异步复位，那么该异步复位应该由如图 8-4 所示的同步器所驱动。

为什么异步复位要由同步器来驱动呢?

当发出复位信号时，没有确定的方法知道复位信号的产生与时钟的关系。一

些寄存器可能先接收到时钟信号，另一些寄存器却先接收到发出的复位信号，这将会导致混合的寄存器状态。如果这是一个短暂的复位，它也许就被完全忽略了。

如图 8-4 所示同步器的电路可以消除所有上述问题。

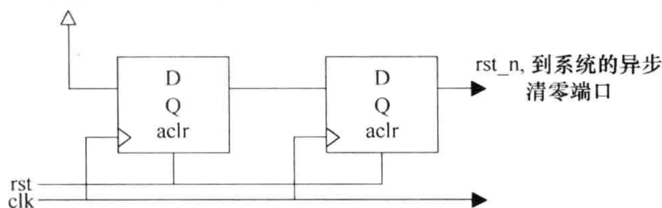


图 8-4　用于异步复位的同步器

8.4.2.1　时钟网络资源

器件范围内的全局时钟布线资源和专用时钟输入都由 FPGA 提供。只要 FPGA 器件中还有低偏斜、高扇出的专用布线资源可用，设计师就应该使用。

设计师应该把设计中时钟的个数限制在所使用 FPGA 器件中可用的专用全局时钟的个数范围内。若不使用全局布线的时钟，来自多处的时钟信号有可能造成器件内部时钟的相位偏差，从而导致时序问题。

使用组合逻辑产生内部时钟会增加时钟线上的延迟。在某些场合，这种时钟线上的延迟会导致时钟的偏斜大于两个寄存器之间的数据路径延迟。如果时钟偏斜大于数据延迟，那么这个设计将不能正确地运行。

8.4.3　专用硬件组件

所有的 FPGA 厂商都为用户提供厂商自定义的参数化组件库，库中的每个组件都能高效地完成某个特定功能。然而，只要在 RTL 代码中实例引用了这些功能组件，那么该 RTL 代码就被锁定在某一个 FPGA 厂商，甚至可能是某个系列的 FPGA 器件上。这显然降低了设计的可重用性。实例引用了这些功能组件后也可能造成 RTL 仿真速度变慢。行为描述的 RAM 模型可能比 FPGA 厂商提供的参数化 RAM 模型的仿真速度快得多。这是由于 FPGA 厂商提供的模型涵盖了所有可能的使用场合，因而仿真速度可能更慢一些。

在某些应用场合，设计师除了使用这些优化的参数化组件之外，并没有其他更好的选择，因为这些组件是使用该器件某些特定功能的唯一途径。使用参数化组件的场合有很多，例如，构建时钟树就需要使用锁相环（PLL）组件，构建高速串行接口就需要使用收发器组件。使用厂商提供的组件是构造这一类功能模块的通常做法。它们通常能够被其他系列的 FPGA 器件或其他厂商的等效技术组件所替换，对设计的影响极小。这非常类似于使用购买的 IP 核。

若想要使用 FPGA 片内组件，诸如片内 RAM 块和 DSP 组件，只需要实例引用这一类组件即可。若想要自己来编写这一类组件，则必须掌握比 RTL 级别更低层次的语言。

这些由 FPGA 厂商提供的功能组件有若干个待定参数需要选择或设置，通常可以通过窗口菜单和对话框，由设计师参考说明书作出正确的选择或设置。

8.4.3.1　实例引用的模块与自己编写由综合生成模块之间的比较（见图 8-5）

	实例引用	自己编写
优点	容易使用，有图形界面帮助	结构独立
	全硬件特性	仿真简单
缺点	特定的结构	灵巧仔细的手工编写
	仿真需要库文件	依赖计算机辅助设计工具

图 8-5　实例引用的模块与自己编写由综合生成模块之间的比较

8.4.4　低层次设计原语的使用

本节讨论由厂商提供特定的低层次设计模块的使用，例如使用进位链和查找表（LUT）原语来实现设计。

自从 FPGA 发明以来，FPGA 设计师就一直使用这种设计技术。在艰难和遥远的过去，使用低级原语设计是保证通过综合实现设计的唯一方法。多年来 EDA 综合工具变得越来越智能，以至于如今这种风格的设计相对于常规来说是特例。对硬件设计而言，它实际上类似于软件的汇编级编程或者用原理图做设计，唯一稍显麻烦的是必须在 HDL 代码中声明模块的连接关系。

然而为什么这种设计风格并没有完全消失？毕竟它是一种繁冗的设计方法，而现在的综合工具是特别智能，使用这些低级原语还会减少设计模块可重用的能力。

在某些情况下，优秀的设计师还是要比综合器聪明一些。让我以加法器设计为例来说明这个问题：综合器通常会重新构造算术结构，一有机会就简化加法器进位链的逻辑。这个简化过程是启发式的，有时会产生次优的分组。如果设计师考虑目标硬件，并依此构建 HDL 代码，就能确保获得可能的最大密度。使用低级原语设计意图十分明确，方法可行，并与周围的逻辑无关。然而，要使这种设计方法发挥作用，设计师需要对加法器按位分片，用以清楚地区分那些是想要的进位输入和想要的进位输出信号。

建议尽量避免使用这些低级原语，除非在最终设计中性能或者面积有问题。如使用标准 RTL 编码技术的设计不能达到设计需要的性能，再考虑使用低级原语来达到设计目标。设计师可以建立起由低级原语构建的自有模块库，例如优化的三进制加法器或者 CRC（循环冗余校验）模块。然而必须意识到的是这些模

块仅仅能在这个 FPGA 厂商的产品中可重用，甚至在某些情况下仅仅能在特定的 FPGA 器件系列中可重用。

8.4.5　亚稳态的管理

如果数据在输入寄存器时违反了寄存器建立时间和/或保持时间的要求，那么寄存器的输出可能会进入亚稳态。在这种状态下，寄存器的输出值会在高电平状态和低电平状态之间振荡。如果这个振荡的值在整个电路中传播，其他的寄存器可能会锁存这个错误的值，从而引发系统错误。

当数据信号在时钟域不相关的两套电路间进行传输的时候，通常会发生亚稳态问题。

为了避免潜在的亚稳态问题，从外时钟域输入的异步信号，在输入本时钟域后，先通过两到三个由本地时钟触发的寄存器输出后再使用是一个好习惯（见图 8-6）。

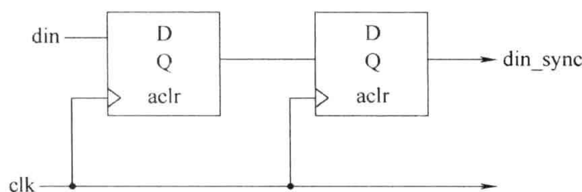


图 8-6　两个寄存器组成的同步器

8.5　编写高效的 HDL 代码

编写 RTL 代码高效率的最重要的原则是分而治之。为了便于找到问题之所在，应尽量将设计分为多个相互之间没有关联的小模块。预先估计设计中可能有问题的地方，特别是总线接口，从那里着手开始设计。系统设计应该搞成这样：即使所有的模块都还尚未出现，设计师还是能够运行单个模块，并对其进行测试。系统设计除了有助于开发过程的早期工作外，当设计中有些模块正在被修改或者甚至还没有时，设计师仍可使设计取得进展。

遵循良好的同步设计方法。ASIC 设计中只要严格地控制时延，使用异步设计是允许的。而在 FPGA 设计中，绝对不允许使用异步设计，因为即使严格地控制时延，也很容易造成麻烦。在设计中采用流水线技术，同时寄存所有的端口，可以带来几个好处。首先，可以将组合逻辑划分为更容易综合的部分。由于 FPGA 验证工具能够容易地访问寄存器的输入和输出，所以流水线技术使调试更加容易。最后，通过寄存器位置的改变，流水线技术能让性能优化有更多的选择。

8.5.1 什么是最好的硬件设计语言

基于本书的目的，我们仅考虑有 IEEE 标准的硬件描述语言（HDL），如 VHDL，Verilog 和 SystemVerilog。

在遥远的过去，有许多种面向可编程逻辑器件（PLD）的硬件描述语言（HDL）。其中一些语言由 FPGA 厂商开发。在 IEEE 批准 Verilog 和 VHDL 标准后，这两种语言就很快就占领了 ASIC 设计市场，并且在 FPGA 市场日益普及推广。Verilog，包括 System Verilog，和 VHDL 允许用户只使用一种语言就能描述设计实现，还能描述用于仿真测试的激励信号，因此具有很大的优势。如今，旧的 PLD 语言已经完全被 Verilog 和 VHDL 代替了。

那么，对 FPGA 硬件设计而言，哪一种硬件描述语言更好呢?

其实最好的硬件描述语言并不存在。所有符合 IEEE 标准的硬件描述语言都有各自的优缺点。

VHDL 通常比 Verilog 冗长，但通常却也有更丰富的特征。VHDL 有强大的类型检查机制，这样就不容易犯糊涂的错误。

Verilog 语言风格简练但类型定义宽松。

总而言之，Verilog 和 VHDL 都能很好地完成 FPGA 设计。究竟选择哪一种硬件描述语言主要由个人偏好所决定。其中的关键因素是当选择一种硬件描述语言的时候，确信已经完全理解了这门语言。由于这两种语言都有很多不明显的语义规则，因而必须仔细研读所选择语言的详细资料。

学习硬件描述语言的起点最好是购买一份 HDL 的 IEEE 标准。虽然 IEEE 标准读起来枯燥乏味，但包含了 HDL 设计参考书经常忽略的细节。

因特网上有从白皮书到 HDL 编码训练课程的丰富材料。这些材料有益于对硬件描述语言产生感性认识，也有助于建立起这门语言的基础知识。众多技术培训中心、本地高等院校、EDA 厂商或 FPGA 厂商都会提供的优质 HDL 课程，建议付费参加。授课老师往往会讲授书本上没有覆盖的丰富信息，而且手把手的实验会给你一些工具使用方面的经验，将来可用于设计创新。

8.5.1.1 在设计中混合使用由不同语言编写的模块

市面上大多数 EDA 综合工具支持由不同 HDL 编写的模块混合而成的设计。然而这样做会面临一些很难克服的问题，除非别无选择，建议在设计中最好不要混用由不同语言编写的模块。

什么时候才不得不混用不同的设计语言，而没有其他办法呢?

1. 购买的 IP 是使用另一种 HDL 编写的，与公司现有的规定不同。
2. 重用由别的公司完成的设计，其模块是用另外一种 HDL 编写的。

如果设计团队中确有一位“天才”，他习惯使用与公司规定不同的 HDL，但

这不能成为混用不同设计语言的理由。相反，这个“天才”需要服从公司已有的规定。

混用不同语言做设计可能会遇到什么问题呢?

1. 容易造成设计不可移植。IEEE 标准对不同设计语言的混用没有作出任何规定，因此 EDA 工具建立它们自己的规则，这很有可能导致设计的不可移植性。

2. Verilog 对大小写字母敏感，而 VHDL 却不是这样。如果设计师采用区分大小写字母的命名策略，很有可能把自己置身于雷区。

3. 并不是所有的仿真器都支持设计语言的混用。绝大部分主要的 EDA 仿真工具支持设计语言的混用，但是费用比入门版的仿真工具高。

尽管建议避免混用不同的设计语言，但如果使用另一种语言编写的模块中引用的实例或实体有比特类型或向量类型的端口和简单参数类型，那么混用语言所做的设计还是可以正常工作的。

8.5.2 良好的设计习惯

8.5.2.1 代码的说明文档

编写清晰的说明文档，对设计中的主要模块做一些介绍，应当成为设计机构的常规。该文档用来对描述设计的 RTL 代码做一些补充说明。它应该包括设计的方块图和分层次的描述，以解释设计的结构。它还应包括时序细节的描述，例如哪些路径的时序需要异常处理。本书第 12 章时序分析中将阐述时序异常处理的细节。

设计中主要模块的说明文档，例如方块图对设计的重用至关重要。如果不知道哪几个功能模块是需要（外购 IP）重用的模块，那么想通过设计重用来缩短设计周期就不太可能实现。当设计师回顾以往所完成的设计，或培训团队中新加盟的成员来接替设计维护或完成设计模块时，设计文档也是十分有帮助的。

设计模块的 RTL 代码应该具有自明性，例如 RTL 代码中信号的命名规则应当能描述信号是做什么的，如 dram _ ctrl，regfile0，crc32，egress _ buffer 等等。注释应该在整个 RTL 代码中广泛应用以解释代码的功能，例如在设计中标注测试信号或多周期路径，以及说明某些模块的用途。

8.5.2.2 对信号命名规则的建议

创建公司自己的命名规则，并坚持下去!

标准的命名规则需要在整个公司中推行。

命名规则可以提高代码的审核效率。市面上有帮助建立编码规范和强制执行编码标准的 EDA 工具。强烈建议购买 EDA Lint 工具，强制执行公司的编码规范。同时编码规范也应成为与版本控制软件交互作用的一部分。为了通过版本控制工具的审核，所有 RTL 代码必须通过 Lint 工具检查，并获得健康合格证明书。

正如前面的讨论过的那样，所有的端口、信号和变量的命名都应该有意义。

下面是一些标准的命名规则，可以考虑作为信号命名规则的一部分。

“reset”或“rst”：复位信号；

“clock” 或 “clk”：时钟信号；

“clk125 或 clock _ 125”：125 MHz 时钟信号；

“rst125 或 reset125”：125 MHz 时钟信号域的同步复位信号；

后缀为“_ n”：低电平有效的信号或者差分信号的负半部分，例如 we _ n 是一个低电平有效的写入使能信号；

后缀为“_ p”：差分信号正半部分；

前缀为“a”：异步控制信号，例如 aclr 是一个异步清零信号；

前缀“s”：同步控制信号，例如 sload 是一个同步置数信号；

“en or ena”：时钟使能信号；

“_ ack，_ valid，_ wait”：总线控制信号；

使用大写字母：来标识参数，枚举类型变量和常量。

虽然一般常量在综合阶段无关紧要，但是它们对理解逻辑结构非常重要。

总线信号的规则：

确保使用总线顺序的一致性。业界最通用的是最高有效位在左（MSB），最低有效位（LSB）在右，例如［63：0］。

避免在声明语句中省略低有效位，例如［7：3］。这会增加设计模块在连接时产生结构化错误的可能性。

然而，忽略未使用的最高位却没有问题，例如用［12：0］来代替［15：0］是安全的。这样做有利于减少综合工具的分析时间，同时也可以减少综合工具产生的警告数量。

8.5.2.3 层次结构和设计分割

层次结构对于设计分割来说至关重要，应该精心设计。合适的层次结构有助于发现设计中存在的问题。但层次过多的结构也会使设计难以理解。因此，需要使层次结构保持在一个适中的深度。

平铺的设计实在很难理解，而且会给调试过程带来许多困难。

应该按照功能边界对设计进行分割。这样更容易看出设计的用意。当评判设计的层次分割时，设计文件的层次结构应该与方块图的精神一致，每个文本文件对应一个 Verilog/VHDL 模块。这将提高对设计的理解，并且不会影响 EDA 工具进行的优化。因为除非设计师明确指示，否则综合工具将自由地跨模块边界进行优化。

这样做的好处是有利于子设计的独立仿真。这也使得设计师能对模块进行快速的性能分析。

当根据功能边界进行设计分割时，设计师应该寄存模块所有的输入和输出。这可能会以增加设计的延迟为代价，然而带来的好处通常远远超过付出的代价。在时序收敛时，这种隔离模块的做法能帮大忙，因为关键路径通常包含在一个分割区域内并且能与设计的其余部分独立运行（见图 8-7）。

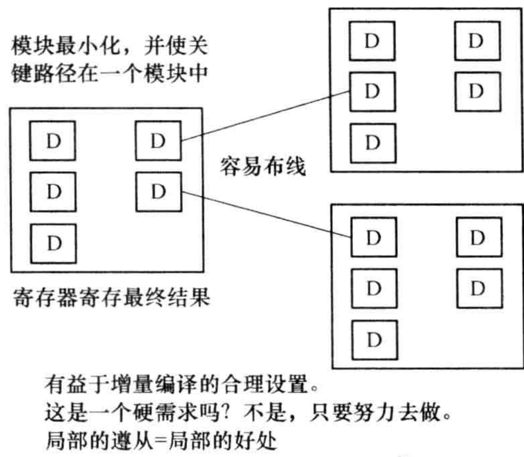


图 8-7　合理的设计分割

最近几年中，这种极其有价值的建议几乎没有被 100% 的设计师所重视。因为寄存模块所有的输入和输出需要对设计进行前期规划。而设计师们常犯的错误是顺着自己的感觉去做设计，“如果以后需要，我再考虑模块端口的寄存”。这种说法极大地低估了不重视前期规划所造成的返工花费。后期任何的延迟改变将会波及到设计的其余部分。

当进行设计分割的时候，设计师必须避免在各分割部分间插入胶合逻辑，如图 8-8 所示。

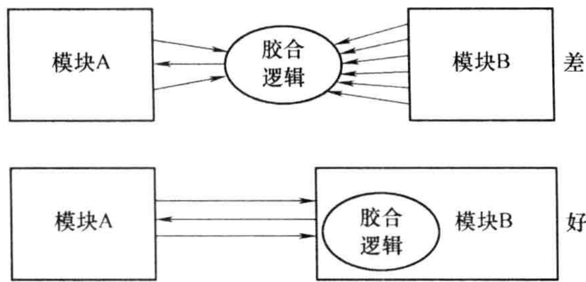


图 8-8　差的分割和好的分割示例

不要在分层边界上使用三态或者双向端口，除非它们一直和器件的 I/O 引脚接口。这是因为在 FPGA 器件的内部没有三态总线，实际硬件的三态功能是用多路复用器来实现的。若在分层模块的边界上使用了三态或者双向端口，则造成模型仿真行为与实际硬件行为的不一致，这很难理解。

建议使用如图 8-9 所示的详细方法来处理这个问题。

```
Input : my_bus_in[16];
Output : my_bus_out[16];
Output : my_bus_oe;
```

图 8-9　处理分割边界上三态的代码示例

良好的设计分割使设计师能够采用分而治之的方法来建立优化的设计模块。

构建模块可以并行开发，当然也可由不同小组开发如图 8-10 所示。

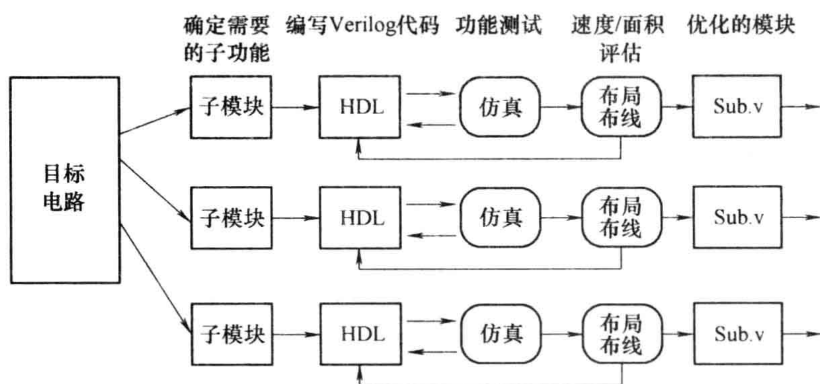


图 8-10　用于 RTL 设计的分而治之方法

花费很少努力就可将这些优化的子模块组合成一个优化的系统（见图 8-11）。

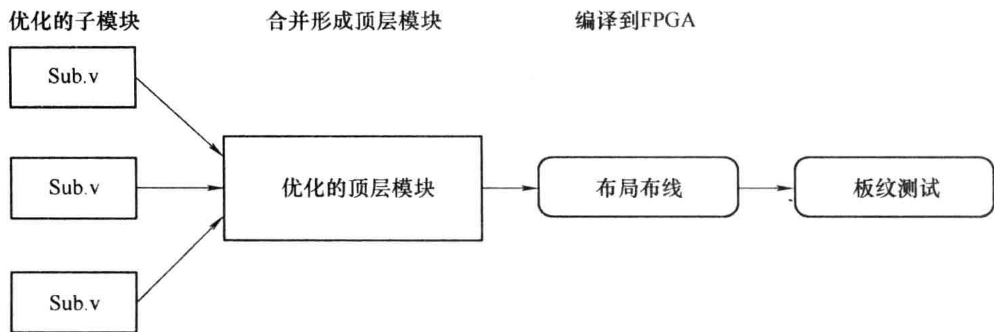


图 8-11　组合子模块来创建优化的设计模块

8.5.2.4　设计的重用

这本书中将用完整的一章专门来讲述设计的重用。在本节中我们将阐述 HDL 编码风格是如何影响设计重用的。

若某设计是采用同步设计方法完成的，而且其层次结构的划分十分合理，则该设计就有重用的可能性。

将某个成功的 FPGA 设计作为一个整体，融入下一代芯片中，加以重复利用的现象十分常见。这也许是为了削减产品成本的缘故。例如，为了下一代的新系统能有更多的功能，设计师先把多个设计融合到一个更大的 FPGA 器件上，然后再移植到专用集成电路（ASIC）上实现。

经过优化的模块通常可以重复利用。但是若在设计中使用了某特定器件系列中专用的原语元件，则该模块可能需要做一些修改后才能重用。

下面列出了哪些要素造就高质量的可重用 FPGA 组件模块：

1. 目标和功能都十分清晰的模块；
2. 能通过参数的设置满足不同用户定制需求的模块；
3. 独立可测的模块；
4. 输入/输出端口拥有信号寄存器的模块。这可以为时序收敛提供保障；
5. 使用标准协议接口的模块；
6. 有清晰明了注释的 RTL 代码模块；
7. 接口信号数目不太多的模块。模块接口信号太多会造成与其他设计模块连接的困难。

高质量的可重用 FPGA 组件模块需要避免：
1. 模块的层次太多；
2. 模块过小；
3. 模块需要的专用信号太多，这样会造成与其他设计模块连接的困难。

8.5.2.5 缩短设计周期的技术

通过仿真和综合技术可以缩短 RTL 代码的设计周期。

多花一些功夫从最底层向上，对子设计进行功能仿真会发现很多问题。等到对整个设计进行全面仿真，或对电路板上运行的芯片进行调试时，发现有问题，就很难找到问题的根源。从最底层起开始调试也许很麻烦，但是在设计的最底层发现并消除错误却要快得多和容易得多。

有许多技术能缩短 RTL 代码的综合时间，以下列举三个技术：

1. 进行面积评估。运行综合工具后可得到关于设计规模的大致数据。设计者也许会问，为什么得到的是大致的而不是精确的结果呢？主要原因有两个。首先，当此模块是与其他模块结合时，综合工具将进行一系列的跨模块的优化。其次，FPGA 布局布线工具还要进行一些优化，例如：用 LUT 包装无关的寄存器、合并存储块。

2. 当子模块的设计马上就要完成的时候，对子模块进行布局布线以确认其性能。如果刚好满足性能需求，那么应该尽量为完整设计的集成留有一些的余量。10% 的余量就行了，当然 15% 的余量更好。

3. 在设计周期的初期，应尽量避免进行任何手动布局或布局规划，而是应通过修改 RTL 源代码来满足性能目标。

有时候这样做不能马上解决问题。当遇到这类情况之一时，应该在设计文档中详细记录，并利用增量设计方法来锁定该模块已经达到的性能。

设计师要尽量减少需要运行的设计迭代次数。因为对大型 FPGA 器件而言，迭代时间是昂贵的开销。在大多数综合工具中，综合器运行时间的长短与设计规模的大小大致成线性关系。综合工具必须完成的工作越艰难，综合花费的时间就越长，与布局布线花费的时间十分类似。

当构建设计时设计师需要记住逻辑块的构造越简单，设计的性能就越好，综合所需的时间就越短。实际上，流水线级数多的设计因其逻辑块的构造更加简单，所以速度性能更好，综合所需的时间更短。

如果设计的逻辑构造过于复杂，那么综合器就不得不反复地分析代码中的逻辑功能，解析出复杂的逻辑构造，从而造成综合时间过长。

8.5.2.6 针对调试的设计

在第 13 章（系统在线调试）中，我们将更深入地讨论本话题。本节将先介绍一些在 RTL 代码级用于提高系统在线调试能力的设计技术。

1. 用寄存器保存想要观察的芯片内部信号。这样做的话，这些信号就不太可能被综合器优化掉。

2. 为便于调试，把设计划分成多个层次。例如，如果设计师想要观察某个接口，可以将该接口配置在器件边沿的输入/输出引脚上，以便于监测。

3. 编写从最终设计中能很容易提取到的测试模块。

4. 确保 FPGA 器件中还有未使用的存储器资源和逻辑资源，以便配置嵌入式逻辑分析仪，用于片内逻辑电路的调试。

5. 保留一些空闲引脚，以备调试信号访问之用。

8.5.3 可综合的 HDL

开发大多数硬件描述语言（HDL）的目的原本只是为了功能仿真，而不是为了将代码转换成电路。因此，用 HDL 描述电路的功能十分容易，但这个功能却不能用硬件可靠地实现。设计师必须意识到许多综合工具仍会把有问题的代码也转换成电路网表，而网表的运行结果与代码的功能仿真结果很可能不一致。本节将不举例介绍那些存在问题的代码，而是建议设计师花钱参加 RTL 编码培训课程或花时间认真读书学习。Verilog 和 VHDL 标准中有一个可综合子集，所有的综合工具都能理解用该子集编写的 RTL 代码。采用这个子集编写的 RTL 代码，综合后的电路运行结果将与 RTL 代码的功能仿真结果完全相同。设计师要学习并严格遵循这个可综合的子集。

下面列出了设计可综合模块的三条指导原则：

1. 在描述可综合的设计时，头脑中要有明确的硬件意识。也就是说，用逻辑门和寄存器确切定义电路的功能。

2. 清楚地知道目标器件的局限性。

3. 当设计成功通过综合后，检查并消除综合工具中出现的警告信息。

8.5.3.1 编码风格

在创建设计时首先需要明确将要进行的设计是结构设计还是行为设计？

在设计实践中，设计师将使用也应该使用结构的和行为的两种编码风格。老

学院派的FPGA设计师将会告诉你必须要使用高度结构化的设计以保证设计的实现和性能。实际上，这样做仅对那些推进性能指标的设计是正确的。如果真有此类情况的话，也仅仅是设计中很少的一部分。

顶层模块总是用线网连，把许多个子实例集合在一起。

大部分子模块采用行为化的风格来实现核心功能。

建议设计师从推荐的可综合编码规范中使用最简洁的语言结构描述设计。这样做使得设计的功能更容易理解。

编写的代码行数越少，需要调试的代码行数也就越少。这是一条编码的常识。

只有在必需的时候才可以实例引用基本原语元件。为了满足性能需求或者使用FPGA器件的某些独特功能，例如使用输入/输出（I/O）原语元件，收发器模块等，才有这样做的必要。

8.5.3.2　常用Verilog编码指导原则

本书不准备广泛地讲解Verilog编码的指导原则，只提出五条最重要的编码建议。

1. 购买一本Verilog RTL编码的参考书或者一份IEEE Verilog标准。
2. 理解非阻塞赋值（ < = ）和阻塞赋值之间（ = ）的差异。

   在组合逻辑建模中使用阻塞赋值（“ = ”），

   在跳变沿触发的always模块中使用非阻塞赋值（“ < = ”），

   除了以下两种情况：

   例外1：给临时变量赋值；

   例外2：给先写后读的RAM赋值。
3. 注意表达式的位宽。

   16比特向量可以直接赋值给8比特向量。

表达式的内容能改变操作数的位宽，即操作数的精度。

4. 注意表达式的符号

   在复杂的表达式中，只要有一个无符号操作数就能够强制改变所有操作数的符号，例如unsigned _ a + signed _ b + signed _ c。
5. 注意隐含的线网声明。

8.5.3.3　常用的VHDL编码指导原则

与上一小节一样，本书不准备广泛地讨论VHDL编码的指导原则，只是提出四条最重要的建议。

1. 购买一本VHDL RTL编码的参考书或者一份IEEE VHDL标准。
2. 标准封装库的合理使用。

   使用时钟信号的上升沿和下降沿作为跳变沿触发条件（ieee. std _ logic _

1164)；用 ieee. numeric _ std 和 ieee. numeric _ bit 程序包作为无符号和有符号类型数据操作的算术运算库。

3. 不要在 case 语句的条件中使用“X”、“U”、“Z”、“ - ” 这些元值。

在 VHDL 语言中操作符“ = ”需要精确匹配。

特别需要注意的是“X” 和“ - ” 的行为与无关项不同。

4. 用实际的动态范围来约束整数的子类型，例如用 integer range 7 - 0 来约束整数范围。这样做可以优化整数的位宽，从而减少硬件的开销。

8. 5. 3. 4 提高性能的设计方法

FPGA 设计达到最快时钟性能的主要做法是采用流水线技术。请记住这一点，无论寄存器使用与否，它们都存在于 FPGA 单元的结构之中，并不因为你没有使用寄存器可以使 FPGA 变小。

设计师应根据说明书上列出的查找表个数和寄存器延迟参数，选择一款逻辑规模和速度性能都符合设计需求的 FPGA，作为实现设计的目标器件。设计师在设计所有子模块的过程中应该维持这个既定的目标器件。

在综合工具和物理综合工具中有高级设置，可以使用诸如寄存器重定时等技术来提高性能。这有利于修正设计中少量的长路径。然而，在 RTL 代码中进行手动修正，既可以保证设计的性能，又可以减少编译时间，而且可使设计模块可重用。即使升级到更新版本的 FPGA 设计软件，这种方法也同样可以保证设计模块的实现。

图 8-12 展示了一个在寄存器间具有两层逻辑的设计。

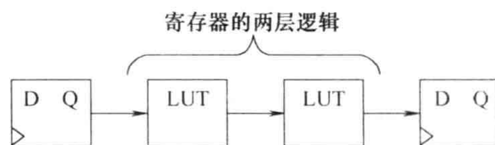


图 8-12 寄存器间的两级 LUT

尽可能地使设计流水线化。图 8-13 展示了如何添加流水线级帮助布局布线达到性能要求。如果数据路径（见图 8-13）的输入和输出引脚被分别安置在这条数据路径的两端，从而造成数据路径贯穿整个芯片，在这条路径上“无用的”寄存器则可以被用于分拆这个长的布线延迟，使得设计满足时钟要求。

时序余量

设计子模块的时候，设计师总是应该提前考虑系统的时序收敛问题。单独编译每个子设计，并使用静态时序分析工具监控其时序性能。设计师应该总是为每个子设计的时序需求留有足够的余量。这将为整个设计的融合留有时序余量。

每个子设计单独执行布局时，能够得到最优的布局布线。然而，把许多子设计整合成一个大设计的时候，并不是每一个子设计都能在整个芯片上实现最优的布局布线。设计师应该预估计到整合后的设计将产生 10% ~15% 的速度退化。时序问题的预防比事后再修改系统时序的设计要容易得多。设计师可能不想把自

己置于这样的困境，即设计后期，设计指标的一个小改动就导致时序从勉强满足变成不满足，从而导致产品上市时间的推迟。

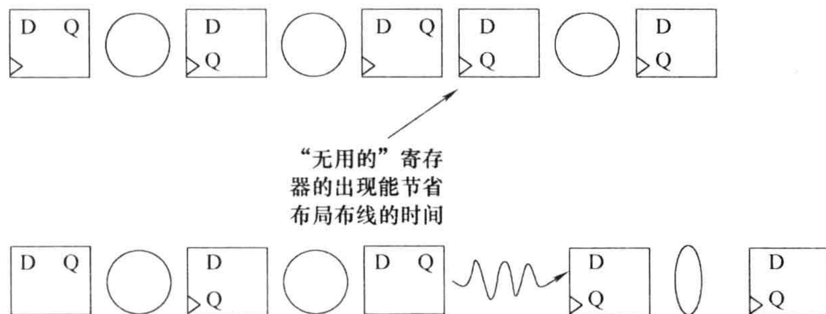


图8-13 使用流水级数来拆分布线延迟

不要相信综合工具的时序估计报告。布局对时序有很大的影响。

子设计往往比较小，所以完成真实布局布线后的时序仿真不需要花太多的运行时间。

8.5.3.5 针对面积优化的设计过程

当编写RTL代码的时候，琢磨一下正在构建的逻辑是什么电路。例如，想用一个加法器还是两个加法器？是否能编写RTL代码只用一个加法器来实现这个电路?

熟悉目标FPGA器件的逻辑结构。寄存器上有什么控制信号可用？查找表（LUT）的构造是怎样的？是4输入的LUT，还是6输入的LUT？

查看综合报告，对所使用的逻辑有一个基本准确的估计。大部分综合工具详细地列出各个层次的资源利用情况。这有助于核定是否有些模块比预期消耗了更多的逻辑资源。

对更小的设计模块，应该使用网表查看工具来分析优化情况，例如电路中有一个加法器还是两个加法器，等等。

如果在设计中存在很慢的逻辑，那么就可以考虑使用时分复用提高资源使用效率。这种方法在DSP设计中很常见，DSP设计中有限脉冲响应滤波器（FIR)，其运行速率可以选择2倍或者4倍，以节约资源。

当检查设计的时候，经常会发现完全相同的寄存器和逻辑块。这是由于几个模块的功能完全相同而造成的。尽管少量的电路复制可能有助于提高运行速度，但是去除这些重复的电路，可以节省很多的面积。如果设计师发现可以节省大量的面积，这表明设计层次的划分做得很差。此时，应该考虑为设计专门构造一个层面，用于放置公共模块。

8.5.3.6　综合工具的设置

为了优化设计使其达到规定的技术指标，所有综合工具都包含很多设置选项。尽管这些设置非常有效，然而却并不能保证在将来的 EDA 工具版本中也有完全相同的效果。使用综合器的高级设置，实际上就消除了 RTL 代码可重用的保障。因此，不管 EDA 综合工具的广告资料如何吹嘘，我们还是建议使用默认的设置来做综合。尽量通过修改 RTL 代码做综合优化，以确保设计的可重用性。若为达到设计目标，不得不使用某项设置，则应该在该模块的文档中对此做完整的记录并加以说明。

8.5.3.7　FPGA 设计块的推断

随机访问存储器（RAM）:

大部分综合工具能够根据代码推断出可进行基本单读单写操作的 RAM 块。

有些综合工具还能够根据代码推断出真正的双端口 RAM 块。

综合工具不能根据代码推断出 FPGA 器件内嵌 RAM 块的所有高级功能。然而，可以通过在 RTL 代码的注释中添加综合属性（指令）或通过实例引用参数化 RAM（原语）组件来实现这些功能。

当编写访问 RAM 组件的 RTL 代码时，设计师应该清楚自己编写的可综合 RTL 代码的读/写控制行为必须与 RAM 宏组件的外部读/写控制逻辑完全匹配。

当描述 RAM 模块时，建议根据综合工具提供的 RAM 模板着手设计。设计完成之后，就可以创建自己的 RAM 读写模型库，在后续的设计中可以重复利用该 RAM 模型库。这样做背后的思想是提前解决综合工具中有关 RAM 器件的所有推断问题。当综合时，可以很容易地使用实例引用的 RAM 宏组件来代替代码中的 RAM 和读写控制逻辑 。

应避免在写操作期间做读操作。综合工具需要嵌入额外的逻辑才能实现同时读写的功能。嵌入的逻辑将会使设计的面积增加，并且使设计的速度变慢。

写操作期间进行读操作的行为

若对同一地址同时进行读/写操作，则读取的是旧数据还是新数据呢？这取决于 HDL 代码的编写。

图 8-14 展示了一段代码的细节，该段代码表明若对同一地址同时进行读写操作，从 RAM 中读取的是新写入的数据。

图 8-15 展示了另一段代码的细节，该段代码表明若对同一地址同时进行读写操作，从 RAM 中读取的是原来保存的旧数据。

图 8-16 展示了一段代码的细节，该段代码对 RAM 进行初始化。

只读存储器

EDA 综合工具能够检测到可以用存储器块中的 ROM 组件实现的寄存器组和逻辑。图 8-17 展示如何在代码中使用 case 语句，并用寄存器保存输出，从而描

述 ROM 的行为。

```
always@(posedge clk) begin
    if(we) ram[addr] = data; //阻塞赋值写入
    q <= ram[addr]; // 如果 we ==1'b1,q 读入新数据
end
```

图 8-14　对同一地址同时进行读/写操作，读取的是新写入的数据

```
always@(posedge clk) begin
    if(we) ram[addr] <= data; // 非阻塞赋值写入
    q <= ram[addr]; // q读地址addr中的旧的数据
end
```

图 8-15　对同一地址同时进行读/写操作，读取的是原来保存的旧数据

```
-- 把 RAM 初始化为全1
Signal my_ram : ram_t : = (other => '1' )

//把 RAM 初始化为全1
ram[31: 0] ram[0 :15];
initial
begin
for (i = 0 ; i<16; i=i+1) ram[i ]=1;
end
```

图 8-16　把 RAM 的内容初始化为全 1

```
always @ (posedge clock)
begin
case (address)
8'b00000000: data_out = 6' b101111;
8'b00000001: data_out = 6' b110111;
……
8' b11111110 : data_out = 6' b000001;
8' b11111111 : data_out = 6' b101010;
endcase
end
```

图 8-17　描述 ROM 的行为

有限状态机

编写有限状态机的时候，应该总是使用异步复位。否则，综合工具将猜测复位状态，从而可能导致设计的功能问题（见图 8-18）。

在 VHDL 语言中，从枚举类型的信号/变量可以得到有限状态机（见图 8-19）。

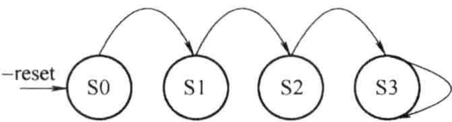


图 8-18　有限状态机

```
type state_type is (S0, S1, S2, S3);
signal my_ fsm : state_type;
State names based on the enum names
```

图 8-19　使用 VHDL 枚举类型来表示有限状态机

在用 Verilog 语言编写的程序中，我们可以从具有以下四个属性的变量推断出程序中有 FSM（有限状态机）存在：

1. 以常量表达式或者模块参数被赋值的变量；
2. 未被声明为输出端口的变量或者未被用于端口连接的变量；
3. 作为整体被引用或者被赋值的变量；

4. 状态名是以二进制表示的数值，或者是以参数表示的数值；

下面是一个 Verilog 有限状态机（FSM）示例，其细节如图 8-20 所示。

描述有限状态机时，一定要指定复位状态，不能忽略：

在 VHDL 中，使用 STATE_ TYPE’ FIRST 进行复位。

在 Verilog 中，复位时把状态值置为 0 或置为最小值。

状态机编码风格

大部分 FPGA 综合工具都认可默认的状态机编码风格。

独热编码常被用于由 FPGA 器件实现的 FSM，这是由 FPGA 的特殊结构所决定的。FPGA 器件中每个单元的扇入较少，而寄存器却很丰富，这正好符合独热码状态机对硬件结构的需求。

二进制（比特位最少）码或者格雷码通常用于由 CPLD 或者乘积项器件实现的 FSM。这两种器件的结构特点是寄存器较少而扇入却很多，这正好符合这两种编码状态机对硬件结构的需求（见图 8-21）。

```
localparam S0 = 0, S1 = 1, S2 = 2, S3 = 3;
reg[2:0] state_reg;
always @ (posedge clk or negedge reset)
if (~reset)
       state_reg <= S0;
else
      case (state_reg)
            S0: stste_reg <= S1;
            S1: stste_reg <= S2;
            S2: stste_reg <= S3;
            S3: stste_reg <= S3;
Endcase
```

图 8-20 用 Verilog 语言编写的有限状态机

状态	二进制编码	格雷编码	独热编码
Idle	000	000	00001
Fill	001	001	00010
Heat_w	010	011	00100
Wash	011	010	01000
Drain	100	110	10000

图 8-21 状态机编码风格

安全的状态机

由于 FPGA 器件中有丰富的寄存器资源，因此通常使用独热码状态机。然而，n 比特的编码就有 $2^n - n$ 个非法状态。在默认条件下，FPGA 综合工具会把设计师人工编写的安全复位逻辑给优化掉。通常综合工具提供一个安全状态机的选项，在综合工具中可以设置这个选项，也可以在源代码中添加综合属性注释，都能实现安全复位的目的。由于硬件中的噪声和混乱事件很可能导致状态机进入不确定的状态，因此务必使用该选项。

在编写状态机代码时如果没考虑未定义的状态，很可能导致综合后生成的硬件在运行时神秘地死锁。考虑未定义的状态是一种良好的工程编程习惯。

大型复杂的状态机

用嵌入式处理器实现大型复杂的状态机较为理想。

大部分 FPGA 厂商提供的软处理器可以使用“C”编程环境来描述状态机的操作，所以实现大型复杂的状态机十分容易。当使用专门的硬件来实现状态机时，每增加一个状态或状态转移都会增加硬件资源的开销。使用软处理器的优势在于除了存储资源外，不管状态机多么复杂，硬件资源的开销是固定的，只有存储器资源的开销取决于状态机的复杂程度。所谓处理器，其实就是一个有很多状态的状态机。这些状态可以存储在处理器的寄存器组中，也可以保存在处理器可用的内存中。FPGA 厂商提供软处理器的好处在于如果状态机太复杂，而 FPGA 的硬件资源又不够大时，可以通过扩大存储器资源，实例引用硬件规模固定的软处理器核，用 FPGA 实现极其复杂的状态机。

FPGA 厂商提供了详细的软处理器使用说明书，指导用户如何在 FPGA 中使用软处理器来实现复杂的状态控制。

DSP 组件

大多数 FPGA 器件内部都包含有一定数量的专用优化乘法器硬件。

FPGA 综合工具能够识别代码中的乘法操作符“ * ”，并且将其转换为相应的乘法器硬件。

有些 EDA 综合工具还能识别代码中的乘法累加操作和乘加操作表达式，将其转换为专用的 DSP 块，实现相同算术的功能。

此外，有些综合工具还可以将输入/输出寄存器映射到 DSP 块以压缩使用的寄存器数量，提高性能和面积的使用率。

然而，DSP 模块有一些更高级的功能，例如高度流水线模式（high pipeline modes），只有通过厂商提供的原语才能使用，而且必须采用实例引用该模块的方法才行，不能通过综合从代码的算术表达式转换实现。

图 8-22 详细地展示了一段执行乘累加操作的代码，综合后将用专用的 DSP 块实现。

寄存器

FPGA 综合工具从同一个基本的 if-else 模板推断得到寄存器。

在 Verilog 语言中，异步（触发）条件与来自异步控制的时钟是有区别的，见图 8-23 所示。

在 VHDL 语言中，用 rising_ edge（）来表示时钟信号，如图 8-24 所示。

设计师必须首先指定所有的异步条件，异步条件的优先权高于同步条件。

寄存器的二级控制信号

再重复说明一下，必须理解目标硬件。

在有些技术中，FPGA 器件中的寄存器只支持异步清零，只有复位信号从高电平接到地才可以清零，并且不支持异步加载。

```
assign multa = dataa_reg*datab_reg;
assign adder_out = multa_reg+dataout;

always @ (posedge clk or posedge aclr)
 begin
  if (aclr)
    begin
     dataa_reg <= 0;
     datab_reg <= 0;
     multa_reg <= 0;
      dataout <= 0;
    end
   else if (clken)
     begin
      dataa_reg <= dataa;
      datab_reg <= datab;
      multa_reg <= multa;
       dataout <= addr_out;
     end
 end
```

图 8-22　乘累加操作

```
always @ (posedge clk or negedge rst)
begin
        if (~rst) q <= 1' b0;
        else q <= data;
end
```

图 8-23　用 Verilog 描述的寄存器的示例

```
if (rst = '0')   then
        q <= '0';
elseif (rising_edge (clk))   then
        q <= data;
end if ;
```

图 8-24　用 VHDL 描述的寄存器示例

不支持异步加载的寄存器很容易产生竞争冒险，这点和锁存器及组合逻辑类似。

使用寄存器的二级控制信号也会影响布局布线。许多 FPGA 器件对可使用的二级控制资源的数量有限制。以 Altera 的 Stratix 结构为例，由于时钟使能（ena）、同步清零（sclr）、同步置数（sload）被同一个 LAB 中的所有逻辑单元共享，所以如果单个 LAB 范围内有太多的控制信号，将会影响设计的逻辑资源使用率（见图 8-25）。

条件语句

使用嵌套的if-else条件语句综合后生成的是具有固定优先级的2选1多路器树。这种编码风格让设计师可以控制晚到的信号，如图8-26所示，这里“a”是晚到的信号。



图8-25 寄存器二级控制信号综合的优先级顺序

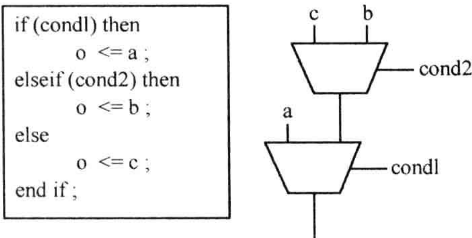


图8-26 多路器

综合上述风格的代码可以得到多路器，但务必注意，当嵌套层次太多时，会显著地增加电路的延时。

如果条件是互斥的，建议使用case语句编写代码。综合后可以生成N选1多路器（见图8-27）。

这种类型的多路器比较容易优化，并且比功能相同有先后次序的多路器的延迟短得多。

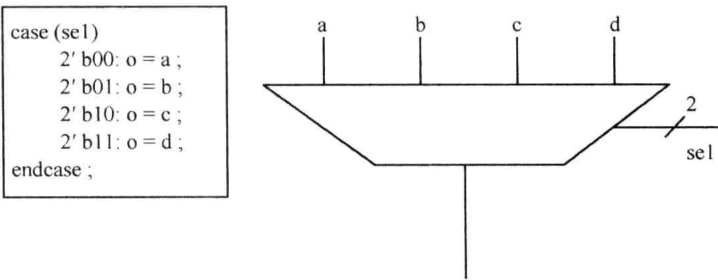


图8-27 N选1多路器

8.6 RTL设计的分析

所有FPGA综合工具都有一套报告设计是否达到性能指标的工具。这些报告还有一个好处，即提供设计架构的细节，这对设计师理解不是由自己创建的设计模块很有帮助。

8.6.1　综合报告

所有的综合工具都能生成详细记录设计关键信息的报告文件。

8.6.1.1　源文件

综合报告会详细地列出设计项目中哪些源文件和库已被综合成网表。这对确保综合器使用的源文件版本的正确性至关重要。

8.6.1.2　综合设置

综合报告会详细地列出综合工具在这次综合中所使用的综合设置选项。该信息对再现综合结果至关重要，应该包含在设计文件中。

8.6.1.3　资源使用信息

综合报告通常按层次列出关于资源使用的信息。该信息对于找到设计中消耗大量 FPGA 资源的区域非常有用。资源使用信息也有助于发现那些不想优化却被优化掉逻辑的位置以及实现的设计与设计师本意不一致的地方。例如，综合报告显示实现乘法操作使用的是查找表（LUT）而不是专用的 DSP 块。

8.6.1.4　状态机

在大多数的综合报告中专门有一小节列出设计中已被识别的所有状态机，其中包括状态机编码的详细信息。该信息有助于发现由编码风格而导致违背设计师意图的状态编码，也有助于发现未被识别的状态机。这些情况会导致非最优的实现，并会影响设计的调试。

8.6.1.5　优化信息

综合报告中还包含对设计的哪些地方已执行优化的信息。对寄存器而言，通常是某些寄存器已被优化掉或被复制的信息。在有些综合工具中会解释进行优化的原因，如寄存器由于没有扇出而被优化掉，或者通过寄存器复制来减少它的扇出。这部分报告还包含连接信息，诸如模块的输入端口或寄存器的输入与地连接。这有助于发现 RTL 代码中可能出现的错误，而且对发现结构化代码连接中的问题特别有用。

8.6.1.6　时序估计

如前所述，综合得到的时序估计并不准确，应该被视为粗略的估计。最好能执行布局布线操作，以获得对设计或子设计时序的更准确估计。

8.6.2　综合警告

设计师应该认真核查综合引擎产生的所有警告，以确保设计正确无误。

综合工具能生成大量严重程度不同的警告。

设计师应该修改代码或者修改综合选项来消除所有的警告。如果该警告不能避免，设计师要完全理解产生该警告的原因，并且要核实这里是否真的存在问

题，然后在模块的文档中加以说明。大多数综合工具提供核查这些警告并且在后序的编译中抑制这些警告的能力。这将大大简化后序编译的核查过程。

然而，建议设计者在最终设计签收之前，完成对全部警告的核查。

8.6.3 电路方块图的浏览

大多数 EDA 综合工具有电路图浏览器选项，可以用于设计分析。浏览器可以为设计创建电路方块图并能提供快速调试 RTL 代码的能力。在大多数情况下，为了方便信号的跟踪和设计代码的调试，这些电路框图和 HDL 源代码之间可以互相切换，交叉探测。

对于那些不是由设计师本人创建，而是从其他设计师那里重用的 RTL 代码，这些工具对增进这些重用的 RTL 代码的理解非常有帮助。它能快速显示设计的结构和设计的数据流。

图 8-28 展示了用 QuartusII® 软件中的 RTL 浏览器看到的电路图。

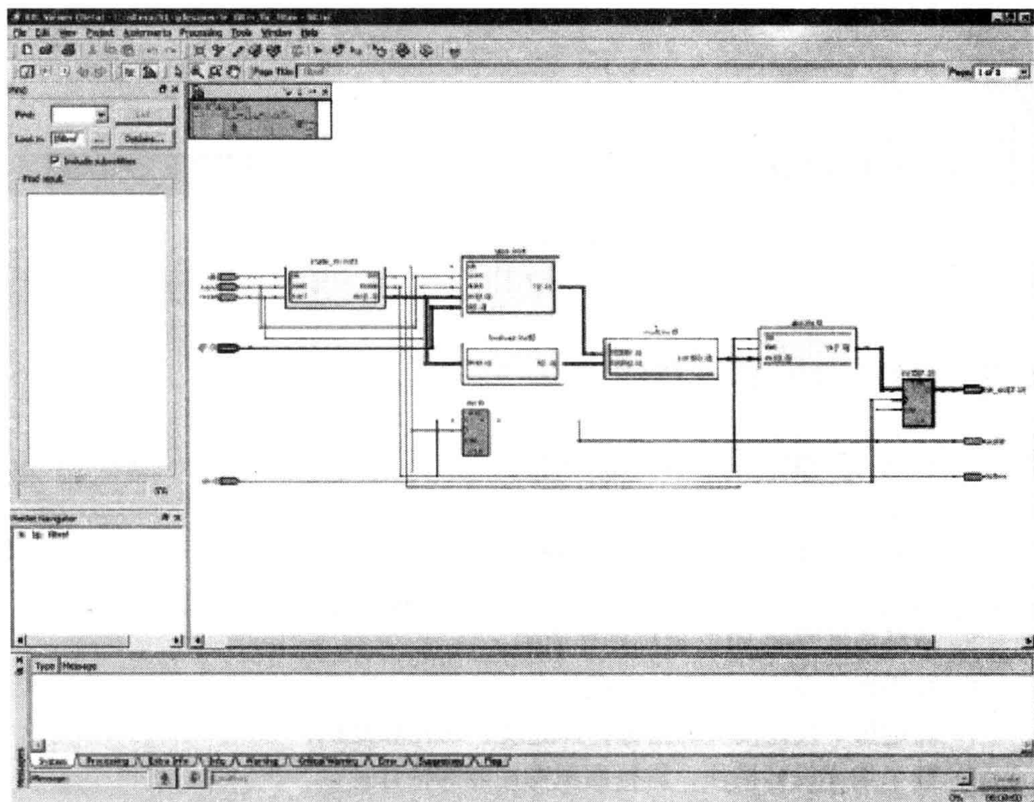


图 8-28 QuartusⅡ的 RTL 浏览器

用 RTL 浏览器可以很容易地观察到状态机的设计，而且可以很容易地确定综合后产生的电路是否符合设计师的期望。

由 Quartus® Ⅱ软件创建的状态机转移图，见图 8-29 所示。

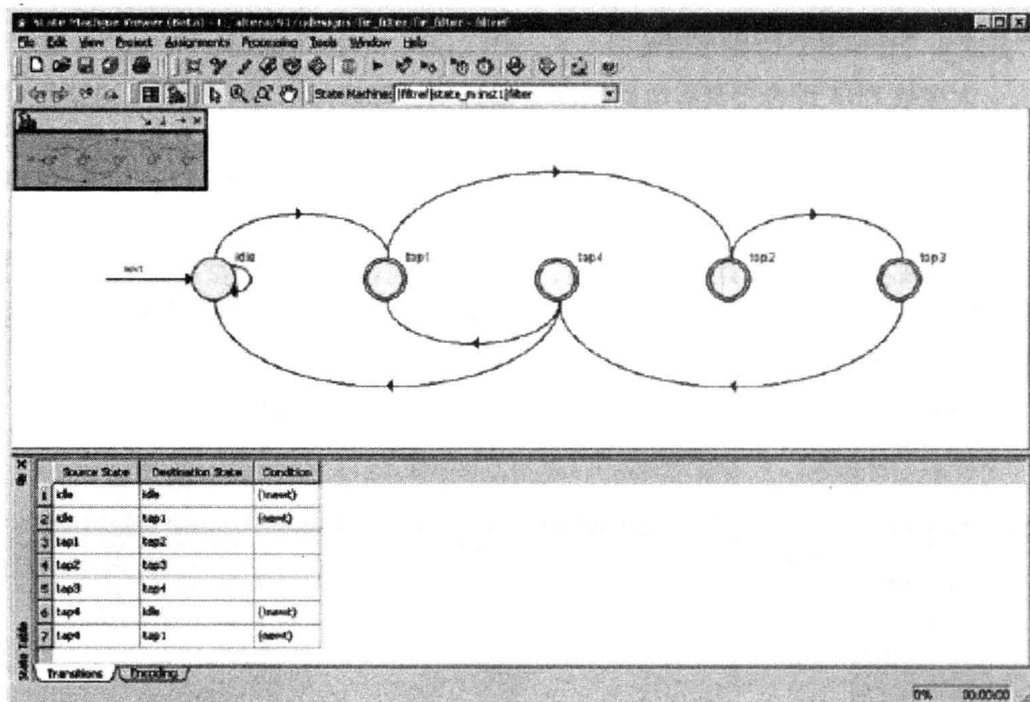


图 8-29　Quartus Ⅱ状态机浏览器

8.7　RTL 设计要点总结

1. 选择一种硬件描述（HDL）语言；
2. 选择 EDA 综合工具；
3. 熟悉所要使用的 FPGA 器件的性能；
4. 创建一个概要的系统设计；
5. 遵循所推荐的 HDL 编码指导原则；
6. 分割设计模块，并逐一实现；
7. 确定每个设计模块的指标，即速度、功耗和面积；
8. 逐个编译设计模块，进行面积和性能的估计；
9. 对每个模块进行仿真；
10. 为每个模块创建一个文档；

11. 消除综合报告中的警告；
12. 组合多个模块合并成为一个完整的工程；
13. 完成整个设计的仿真；
14. 分析完整设计的综合报告；
15. 消除完整设计的警告；
16. 完成整个设计的文档；
17. 对整个设计进行时序收敛工作。

第 9 章　IP 及设计重用

9.1　引言

本书的主要目的是指导设计师如何编写可在 FPGA 器件中重用的设计模块，阐述的主题从编写设计规范起，一直到 RTL 代码的编写和设计验证的整个过程。本章关于 IP 重用的论述和其他章节的内容是互补的。这里着重于讲述 IP 重用的好处，如何决定是由自己设计 IP，还是外购 IP，以及如何对 IP 进行封装以方便使用。

9.2　IP 重用的需求

工业界普遍认同 IP 重用能减少工程量，因此能加快产品的上市时间，从而降低开发费用。

众所周知，很多新产品的设计是在老产品的基础上发展而来的，这实际上就是设计的重用。在大多数情况下，新产品会在老产品的基础上增加一些新的功能，而老产品的设计在新产品的设计中被作为一个整体加以利用。

然而，当需要设计一个全新的产品或者由其他设计小组设计别的产品时，设计的重用并不普遍。

然而，在实际工作中，设计模块却完全可以被别的小组用在其他产品的设计中。

但是，为什么设计的重用依然不多见呢？

主要原因是大多数公司还没有采纳已被开发机构普遍认可的设计可重用方法学。

模块设计工程师通常不会主动地在公司内部推广设计可重用方法学。但他们会是设计可重用方法学的使用者和贡献者。

因此，工程管理部门有责任从高层推广设计可重用方法学。

9.2.1　IP 重用的好处

采用设计重用方法学主要有以下五个好处：

1. 可充分利用现有的投资。每个小组都在做自己的设计，而这些设计的功

能通常基本相同，这样做显然浪费人力。如果能在很多个设计中重复利用功能相同的模块，就可以充分地利用原始设计的投资。

2. 可获得预期的结果。因为现成模块的性能是已知的，所以使用现成模块做设计，可减少不合格设计的数量。当设计模块用另一种 FPGA 技术实现时，如果模块代码是遵循第 8 章建议的 RTL 编码规则编写的，那么在新技术中编译该模块并快速评估其性能就比较容易。这比从头开始编写和验证新的 RTL 设计要快得多。

3. 使工程师能集中精力发挥自己的核心能力。设计中的某些元件可能不是设计师本人精通的领域。若能利用该领域专家设计的模块，则设计师就可以致力于他自己的专长。例如，包处理设计中数据通过以太网接口传输到芯片上。设计工程师可能是一个包处理专家，但并不擅长开发以太网接口。通过重用已有的 10G 以太网接口模块，设计师就能把精力集中在实现包处理功能的核心技术上。

4. 可以最大程度地缩短验证周期。因为重用的设计模块以前早就被验证过，所以只需要作为整个系统验证的一部分，对它们做再次验证。

5. 可以更快地投入市场。将现成的设计模块添加到系统中可能只需要几个小时的操作，而编写复杂的功能模块（如 Interlaken 或 DDR3 存储器等接口模块）并实现可能要花上几个月的时间。

9.2.2　开发可重用设计方法学面临的困难

设计重用不是唾手可得的，需要一定的投入。采用设计重用技术，无论在成本和设计效率方面都能产生巨大的经济效益，然而，公司中每个工程部门在思维习惯上必须有一个重大的转变。

9.2.2.1　工程师的思维习惯

设计重用面临的第一个挑战是让工程师们在思维习惯上做角色的转换，把自己从模块设计师转变为现成模块的使用者。很多公司都患有“非本公司创造综合症（Not-Invented-Here，（NIH） syndrome）”。换言之，公司中有些工程师认为若在设计中利用了别人编写的成熟模块，是很没有面子的，降低了自己的人格和创造的价值。他们希望创建自己的设计而不是使用别人的代码。

此外，有些设计师在创建模块时，他们往往想把这些模块作为自己的知识产权。他们认为与别人分享设计模块也许会减少自己对该模块的所有权。他们也可能担心当其他设计师重用他们设计的模块时，会批评他们的设计。

事实上，最大的障碍在于使设计模块能够重用需要投入极大的努力，而公司方面，却没有给工程师们留足够多的时间做这些繁杂的工作，工程师们也不愿意花自己的功夫为他人作嫁衣。

公司通过建立正式的开发规范来解决面临的难题。接受规范的最初阵痛过去

后，遵守规范就会成为工程师们的一种生活方式，进而他们将会以创建可重用的设计模块为骄傲，就像他们现在创建自己的设计一样。

9.2.2.2　可重用设计模块的推广

推广 IP 是一个难题。工程师们需要知道到哪去找可能对他们设计有用的模块。IP 模块的使用者需要有能力找到有关该 IP 性能指标的说明书，以及如何使用和验证该 IP 的资料。这将消除他们对设计质量的所有顾虑。

同样，工程师们需要知道如何发布自己设计的 IP；在这里发布的意思是指如何使自己设计的 IP 核为其他使用者所知晓。

对采用 IP 重用方法学而言，IP 的推广和确认可能是最大障碍。由于 IP 的使用者没有直接接触该 IP 原来的设计过程，他们需要很多有关的信息与该 IP 封装在一起。这些信息包含文档、验证计划和测试等。

这些问题可以通过一个普通的管理设计重用的网站（例如与版本控制软件关联的维基或 SharePoint 网站）来解决。

9.2.2.3　开发工作

设计重用往往需要花费很长的时间，并付出极大的努力，因此可重用模块的开发代价远高于某个项目中一次性使用的模块。项目进度计划是决定某个模块是否应该进行可重用设计的一个因素。为确保所有项目的进度，重视设计可重用性的公司需要确保所有项目的进度计划均允许关键模块被设计成为可重用的模块。这样做，未来的设计效率将会有更大的提高。

并非每个设计组件都需要重用，因此避免不必要的重用设计至关重要。

确定哪些模块需要做可重用设计是一项很困难的工作。清晰地界定在不同应用环境下都能顺利地重用的模块并非易事。

因此，在编写模块设计说明书时，有必要关联公司内部的其他产品和应用，理解该模块的功能。这些信息可以被用来决定该模块是否应该按可重用方式进行设计，在模块设计说明书中对此应该有明确的表述。

某些小模块（如地址解码器和仲裁器）的设计，最好让系统集成工具来完成。

同样，如果想要设计一个模块，其功能与时序的关系非常密切，对速度性能的要求又十分苛刻，那么很可能在其他 FPGA 器件系列上，甚至同系列的其他 FPGA 器件上都不能实现该设计的重用。所以，对该模块只能做一次性的设计，并不需要遵循所有的设计重用建议。

9.3　设计还是购买

每当需要开发 IP 时，工程经理们总会遇到这样一个问题：究竟在公司内部

自己开发好还是到外面购买好呢?

IP 对满足设计整体性的需求是否十分关键是决定自行设计 IP 的重要因素之一。自行开发的设计模块能更好地控制设计的优化，更能满足未来的特殊需求。如果考虑到这些，设计师就应该考虑自行设计。当然如果能得到源代码，也可以利用其他小组已经开发的设计模块。

同样，如果该 IP 模块是使您的产品在竞争中脱颖而出的特色之一，您肯定有强烈的愿望想彻底理解 RTL 代码的功能并拥有该 IP 源代码的知识产权。

考虑 IP 是自行开发还是外购的另一个因素是成本。设计者需要知道自行开发该 IP 和验证其功能的成本，并与外购的现成解决方案做对比。

然而产品的上市时间可能会迫使设计者去购买 IP。如果设计进度紧迫，且已有的资源已被完全利用，购买 IP 也许可以节约几个月的开发时间。

目标 FPGA 技术的 IP 可用性是另外一个需要考虑之处。从 FPGA 系列的新成员面市起到能在这个新器件上运行的 IP 出现之前，通常需要一段时间。很多小型 FPGA 厂商都会等待主要用户先去测试这些 IP。这会导致时序要求高的 IP 在可用性上有一个延迟。第一个使用新 IP 的用户可能成为新技术下验证 IP 的试验者，对这个用户来说会有一定的风险。当然在领先技术中率先使用该 IP 也会有好处，设计者可能会在竞争中拔得头筹。

每当设计者从其他资源中获得设计模块时，总会担心该设计模块的质量，特别是在购买 IP 的时候。

目前还没有关于 IP 质量的产业标准可以帮助设计者选择 IP。过去有过一些倡议，但都未达到能被业界认可和采纳的程度。因此设计者需要依赖于 IP 供应商的声誉，或者详细询问 IP 供应商的验证过程及想要购买的 IP 的运行效果来选择 IP。

究竟自己开发模块省钱，还是外购模块省钱，设计者可以根据上面列出的所有情况做一个比较。

若设计小组对所需的功能领域并不了解，也没有什么经验，则应该当机立断购买 IP。

9.4 构建可重用的 IP

9.4.1 设计说明书

首先，在整个系统的设计说明书中应该明确指出哪几个新模块是待开发的并可以被用于其他设计。这将会影响这几个模块开发计划和技术说明书的编写。

因此，设计说明书中必须把这几个模块明确定义为可重用模块，并应遵循

IP 可重用设计的指导原则进行设计。

当这些可重用模块的设计说明书被审核时，应当包含可能使用这些 IP 的其他小组的审核意见。这样做有三个主要目的。首先增加各个小组对该 IP 的认识程度。其次，在编写技术说明书阶段就使其他设计小组参与进来，他们在该 IP 中就会有既得利益，从而会更愿意在自己的设计中采用这些 IP 模块。最后，其他设计小组可能会提供一些被设计者自身忽略的反馈意见。

9.4.2　实施方法

9.4.2.1　参数化 RTL 代码

使用参数化的 RTL 代码是业界最常见的 IP 开发方法。它是创建和维护可重用模块的最简单方法。下面举几个例子，说明如何使用参数来设置存储器和 FIFO 的数据位宽。

由于参数可以传递不同的常量，所以参数化模块可以为设计提供内置的灵活性；在 Verilog 语言中的参数是 parameters；在 VHDL 语言中的参数是 generics 。

在决定将 IP 的哪几个常量变成可变参数时，设计者应该考虑该 IP 核的可能用途、期望的功能变化范围，从而为每个想要的功能设置相应的配置参数。

在 Verilog 和 VHDL 中都有生成语句可用，生成语句与可重用 IP 中的参数互相配合，可以生成复杂的实例，有效地实现 IP 的重用。生成的实例和模块参数可以被用于消除冗余的逻辑，从而创建灵活的设计。

用“for”循环构成的生成循环可以包括多条语句也可以实例引用多个块。

```
module bus_mux (din,sel,dout);

parameter DAT_WIDTH = 16;
parameter SEL_WIDTH = 3;
parameter TOTAL_DAT = DAT_WIDTH <<SEL_WIDTH;
parameter NUM_WORDS = (1 <<SEL_WIDTH);

input [TOTAL_DAT-1:0]  din;
input [SEL_WIDTH-1:0] sel;
output [DAT_WIDTH-1:0] dout;

genvar i,k;
generate
    for (k = 0;k < DAT_WIDTH;k = k+1)
    begin : out
        wire [NUM_WORDS  1:0] tmp;
        for (i = 0;i<NUM_WORDS; i = i+1)
        begin : mx
            assign tmp [i] = din[k+i*DAT_WIDTH];
        end
        assign dout[k] = tmp[sel];
    end
endgenerate
endmodule
```

图 9-1　Verilog 源文件中参数使用的细节

生成语句可以用来创建参数化逻辑。图 9-1 展示了一个使用带参数的生成语句来产生总线多路器的例子。

在第 8 章 RTL 设计中，可以找到针对可重用 IP 如何编写其 RTL 代码的更详细的指导原则。

第8章的第8.5.2.3节中提供设计分层和分区的指南。第8.5.2.4节则提供了针对设计重用的编码指南。

9.4.2.2 高层次综合

高层次综合对算法探索非常有效，特别在使用Ansi C/C++作为设计输入的数字信号处理领域更是如此。这一类工具可以大幅度地减少编写RTL算法代码的时间，从而为软件或系统工程师等新人进入硬件设计行业开启大门。当算法描述与算法模型非常接近或完全一致时，这些工具在算法设计的架构探索阶段表现杰出。由于描述功能所需的'C'代码行数远比描述硬件实现的RTL代码的行数少得多；因此设计效率有显著提高。这些工具也能为把设计移植到不同FPGA系列的器件上提供更多的灵活性。在设计的最高层次上，代码并非是为某一种目标FPGA器件而编写的。

然而，对精心调整优化的质量结果（Quality of Results，QoR）而言，通过高层次综合工具得到的解决方案往往不是最佳的解决方案，这是它们的主要缺点。综合后的面积效率不高，或有些性能不能达标。近年来，在某些类别DSP应用中，这些工具在结果质量（QoR）方面取得了良好的进展。总而言之，在创建性能要求不高的DSP IP时可以考虑使用这些工具。

除了C/C++工具外，还有另一类基于模型的设计工具。这些工具通过Simulink提供与MATLAB环境的接口。同样，这些工具大多针对DSP应用。它们已在一个较小的应用领域中取得了成功，大多数是在调制解调器设计和一些军事领域的应用。换句话说，在这些应用领域中创建IP时，可以考虑使用这类工具。

9.4.2.3 IP生成器

IP生成器使用C++、Perl或者其他高级语言编写而成，根据终端用户的参数设置来动态地生成RTL代码。这些生成器可根据所选的参数把几个RTL设计模块的整合在一起。

FPGA厂商常用这种技术为他们的客户群提供复杂的IP。

IP生成器解析用户指定的所有参数，根据设计说明书生成HDL代码。

它们适合用于复杂的参数组合，复杂的合法性检查和高级的算术运算处理。

IP生成器的缺点在于必须具有软件编程技能才能用它生成IP。

9.4.3 标准接口的使用

建议所有的IP采用通用的标准接口。使用标准接口可以简化设计的构造，便于功能模块的互联和管理。使用IP标准接口有以下六个好处：

1. 确保由不同设计小组或厂商设计的IP组件的兼容性。
2. 可以很快地把多个IP组成完整的系统。由于IP的使用者知道接口信号是如何操作的，所以与设计模块的接口逻辑可显著地简化。

3. 为系统整合开启了使用设计自动化工具的大门。

4. 由于设计团队的个别成员就能编写并测试他们自己设计的模块，采用标准接口使基于团队的设计变得简单方便。通过对常用接口协议的理解，每个团队成员都知道如何与常用标准接口模块连接。因此，把多个设计模块组成完整的系统变得十分简便。

5. 标准接口使得 IP 能即插即用。

6. 标准接口使 IP 的稳定性得以提高。接口协议说明书详细地描述了接口信号的操作，而且 IP 核的接口信号可对照说明书进行操作和验证。

如今市面上有各种标准的接口。在 FPGA 和 ASIC 设计中采用最广泛的接口标准是 ARM 公司的 AMBA（AXI，AHB 和 APB），Altera 公司的 Avalon（MM 和 ST），OCP-IP 公司的 OCP 和 OpenCores 公司的 Wishbone。

选择标准的接口协议时，设计师必须确保该 IP 的基本结构可以植入作为设计目标的 FPGA 中。这里所提及的 IP 基本结构指的是在目标 FPGA 的组件库中设计师可以找到该 IP 的组件模块，该 IP 必须符合设计所要求的标准接口协议，而且关于该协议的说明书是确实可靠的。IP 应该包含两个部分：

1）可综合为最终设计的电路结构部分；

2）验证该 IP 的测试模型，如总线功能模型。

标准接口必须容易理解，简明扼要，标准接口在硬件实现时不应该产生性能和面积的损失。该标准接口必须支持设计者所有的应用需求。这通常包括基于地址读/写的存储器映射接口，典型的接口有主从连接接口，支持单向数据流的点对点接口，包括多路选择的流、包和 DSP 数据。

总而言之，使用标准接口协议确实是设计可重用策略的核心。

9.5 IP 组件库软件包

IP 组件库软件包是 FPGA 开发环境中的一个实用工具箱，其中包括由许多 IP 核构成的组件库，以及选用和配置这些 IP 组件的配套工具。

好的 IP 组件软件包应该使设计者垂手即可取得工具箱中的每个工具，很容易找到想要的 IP 组件，安装和维护每个 IP 组件应该非常容易。

设计师可以从公司的可重用 IP 库中找到所需的 IP，或者从用户工作站或设计环境中发起安装某 IP 组件的请求。若某 IP 组件需要安装，则建议设计师通过现成的商业工具来完成该 IP 组件的安装，例如通过 IP 安装界面，也可以用 WinZip 或类似的程序创建一个自解压的可执行文件来完成 IP 组件的安装。

一个 IP 组件包的最低要求如下：

1. IP 核，即满足设计需求的组件；

2. 时序约束和所有的位置约束；

3. 仿真模型；

4. 使用说明书，应该包含该IP核的用户手册以及所有的勘误表。关于这部分内容的细节见9.5.1节；

5. 用户接口；

6. 与设计师想使用的任何系统集成工具兼容。

9.5.1　IP说明书

如前所述，IP说明书应包括用户手册和勘误表。它还应当包含详细记录IP核和文档变化历史的版本控制文档。IP核的版本必须与其本身及说明书保持一致。

虽然对某个IP核而言，其设计功能很可能是独一无二的，但所有IP核说明书的格式却要求完全一致。这包括使用说明书和具有自说明性的RTL代码格式。

说明书应当包括一个用于展示该IP如何与其他模块进行连接的设计举例或者测试平台。在理想情况下，这个例子或者测试平台可以用于演示该IP的功能。

设计的文件结构必须和所有其他的IP相同，并且信号的命名规则必须遵循公司的编码指导原则。

对于参数化IP组件，应该有设置参数的提示。

9.5.2　用户接口

在公司内部推广IP，让其他设计师都知道有这些IP存在，可用于设计，最常用方法是提供这些IP的RTL代码，以及提供如何在设计中使用它们的说明书。虽然这样做有效果，但对最终用户而言，真正懂得如何使用这些别人编写的IP是一件十分困难的工作。

提供IP组件时，必须提供其接口清单，这样做可以帮助用户理解施加在该IP组件上的约束条件。同时，还至少需要提供一个命令行的脚本文件，该文件可以帮助用户将组件的配置参数传递到IP中。在理想的情况下，设计工具应该提供图形用户界面（GUI）帮助用户着手该IP的使用。

建议设计师为自己设计的IP提供一个简单的图形用户界面（GUI）和一个接口脚本。

用户应能通过这个简单的图形用户界面（GUI），为选用的IP设置所需的参数、设置约束条件，并能验证用户选用的IP参数和约束是否合理。

这种类型的接口可以帮助设计师理解该IP的功能，为设计模块生成正确的验证文件和脚本，同时提供一个链接，方便设计者找到有关该IP的使用说明书。

这种类型的接口，设计者经常可以在IP上见到，它们是由FPGA厂商提供

的，在大多数情况下，是由其他 IP 供应商提供的。

配置 IP 的图形用户界面（GUI）应尽量简明扼要，只要能向设计者展示配置界面能对哪些参数进行设置，并允许他们进行设置即可。

Altera 公司开发工具中元件编辑器的图形用户界面（GUI）如图 9-2 所示。

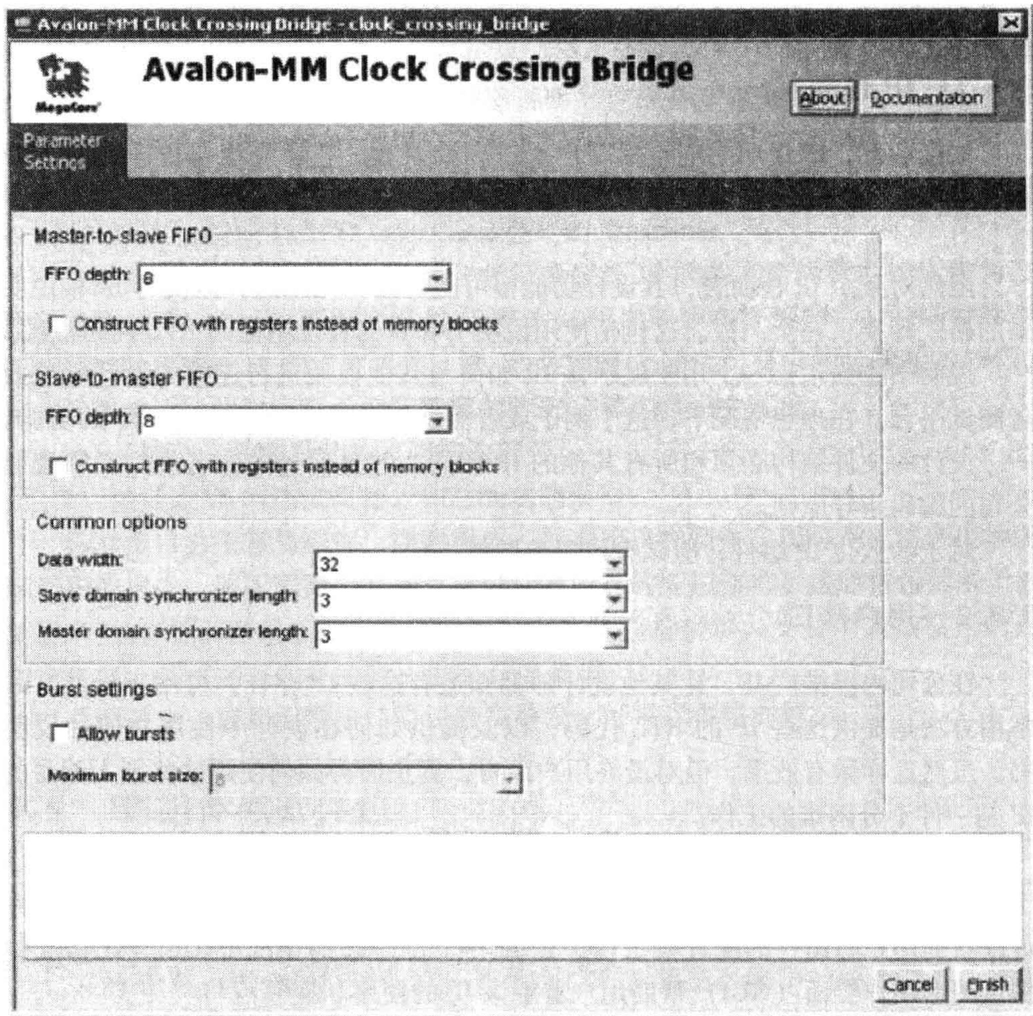


图 9-2　Quartus Ⅱ开发工具中元件编辑器配置 IP 的图形用户界面（GUI）

若设计师的软件编程能力还行，则用 Tcl / Tk 或 Java 编程语言创建一个类似的图形用户界面（GUI）并不困难。

若设计师的软件编程能力不行，则可以采用 FPGA 厂商提供的 IP 图形用户界面开发工具，但这样做必须使用 FPGA 开发工具。

9.5.3 与系统集成工具的兼容性

标准的设计输入和设计集成工具能减少设计输入的开销。

系统集成工具可以根据IP模块的互联自动生成HDL代码。由主要FPGA厂商提供的IP集成工具都拥有此项功能。这些系统集成工具能够自动完成RTL设计师不得不处理的比较无聊的设计任务，如地址译码、数据复用、处理器系统中等待状态的产生、动态总线位宽调整、从动方仲裁和模块的直接互联等设计任务。这个能自动完成设计任务的工具类似于软件连接器。这个软件连接器可在主程序外创建一个可执行程序，即选择某个已经过预编译的IP库函数转换为具体的电路网表并生成相应的仿真模块。

系统集成工具，如Altera公司的SOPC Builder，允许设计者根据需求从多种系统模块中选择必要的组件，自动地构造出一个完整的复杂互连系统。这使工程师们能够把低层次的集成细节交给计算机处理，集中精力思考更有价值的系统架构，大大提高设计效率。

在系统设计的架构探索阶段和实现阶段应该使用这种系统集成工具，以便提高设计效率。系统集成工具拥有以下功能：设计模块能即插即用地连接到系统中，选定的架构可以自动快速地转换成RTL代码，无需手动修改仲裁逻辑、位宽适配逻辑、存储器映射等。这些功能使得系统结构的探索简单易行，设计师就能快速地尝试不同的架构。一旦找到了理想的系统架构，设计师就能微调系统中的模块，以满足设计的总体目标。

9.5.4 IP的安全性

设计师从IP供应商处购买的IP通常是经过加密处理的。IP供应商这样做是为了保护IP组件的RTL代码的完整性并防止非授权用户利用他们的IP进行设计。IP供应商和EDA供应商使用的加密策略往往各不相同。然而，从IP使用者的视角来看，设计师所关心的是哪些工具能支持该IP的综合，以及从IP供应商那里得到的仿真模型的质量是否满足设计要求。

工业界中已采取行动提供标准的IP加密方法。IEEE学会已经发布了基于开放模式接口（Open Mode Interface，OMI）的IEEE 1499标准。该标准使RTL代码能够被编译成为一种禁止反向设计的模型格式。这些模型能在与OMI兼容的仿真器中进行仿真。它的好处在于仿真模型和综合模型使用同一个RTL代码。这样做减少了IP供应商的开发工作量。

有些IP供应商愿意提供IP的源代码。这简化了设计流程，但其购买成本通常远高于加密过的RTL代码。

如果设计师打算提供加密的IP，就必须与相关的FPGA厂商合作，以利用它

们的加密工具。

有些 IP 供应商提供模糊化后的 RTL 代码。让信号的命名显得毫无意义，从而使代码难以理解，这为 IP 的 RTL 代码提供了一种有限的安全措施。模糊化使得非授权用户难以对 RTL 代码进行逆设计。但模糊化并不能妨碍设计的编译。

有些 FPGA 厂商允许设计师只提供编译后格式的 IP，而不是 RTL 代码的 IP。举个例子，在布局布线锁定的情况下使用增量的编译方法对设计模块进行编译。这一级别的 IP 可以确保 IP 的性能，又减少了维护 IP 的负担，但 IP 只能应用于某种特定的 FPGA 器件。

设计师可以用多种方法为其他用户提供 IP。大多数公司内部的 IP 重用，提供的是 RTL 代码，而出售的 IP 则需要对其代码进行加密。然而，有些公司在发布关键 IP 模块时，在公司内部也实施加密策略。

由于加密会增加设计流程的复杂性，所以建议只在设计模块的安全性是主要考虑因素时，设计师才对设计模块使用加密或模糊化措施。

9.6　IP 重用的检查清单

1. 购买还是自己设计该项功能?
2. 说明书中是否已明确说明该项设计是可重用的?
3. 选择恰当的 IP 实现方法，换言之，是用 RTL 编码，还是用高层次综合器生成该 IP?
4. 如果自己编写 IP 组件的 RTL 代码，则应遵循 RTL 编码的指导原则。
5. 如果自己编写 IP 组件的 RTL 代码，则应将该 IP 参数化。
6. IP 设计模块应使用标准接口。
7. 必须对 IP 的 RTL 代码做加密或模糊化处理吗?
8. 自己设计的 IP 是否已遵循 IP 封装入库的指导原则?

第 10 章　硬件到软件的接口

10.1　软件接口

应用软件和 RTL 代码之间的主要接口是寄存器地址映射表。在设计过程中，软件、硬件和其他有关人员都需要参考寄存器地址映射表。

这给工程项目中的固件、RTL 代码和硬件验证、以及它们与技术文档的一致性造成了困难。这不仅涉及到公司内部使用的技术文档，还涉及在 IP 开发时提供给最终用户的文档。

正因为如此，为避免重新编写硬件和/或固件，应该尽量减少信息的修改，必须对信息修改严加管理，并将每次信息修改的细节传达给整个设计团队中的每个成员。

10.2　寄存器地址映射表的定义

寄存器地址映射表通常有许多不同的名称：控制和状态寄存器（CSRs）、内存映射寄存器、寄存器文件、寄存器块或寄存器接口。控制和状态寄存器专门用于存放软件与硬件之间相互交流的数据。每个 IP 模块提供一个可映射到软件接口地址的寄存器接口。寄存器地址映射表使得软件编程人员可以看到软/硬件之间的接口，从而进行正确的读/写操作，以便在硬件和软件之间进行有效的通信。

10.3　寄存器地址映射表的使用

正如本章开始提及的那样，在整个设计过程中，多个领域的专业技术人员需要使用寄存器地址映射表，不同专业所要求的数据格式略有差异。

10.3.1　IP 的选择

作为 IP 评选准则的一部分，设计者必须理解软件和硬件如何与该 IP 接口。寄存器地址映射表就是为解决软件如何与 IP 接口而创建的。IP 核的用户使用说明书应该反应这个信息。

10.3.2　软件工程师的接口

为了开发和硬件接口的软件驱动程序，软件工程师需要熟悉寄存器地址映射表。软件工程师总是希望寄存器映射信息以软件头文件的形式出现，在头文件中定义该组件的基地址和寄存器偏移地址（见图 10-1）。

```
#ifndef __ALT_ETH_10G_REGS_H__
#define __ALT_ETH_10G_REGS_H__

#include "alt_types.h"

/* Revision register */
#define ALT_ETH_10G_REV_REG                    0x00
#define IOADDR_ALT_ETH_10G_REV(base)           __IO_CALC_ADDRESS_NATIVE (base, ALT_ETH_10G_REV_REG)
#define IORD_ALT_ETH_10G_REV(base)             IORD_32DIRECT (base, ALT_ETH_10G_REV_REG)

#define ALT_ETH_10G_REV_CORE_REVISION_OFST     (0)
#define ALT_ETH_10G_REV_CORE_REVISION_MSK      (0x0000FFFF)
#define ALT_ETH_10G_REV_USER_REVISION_OFST     (16)
#define ALT_ETH_10G_REV_USER_REVISION_MSK      (0xFFFF0000)

/* Scratch register */
#define ALT_ETH_10G_SCRATCH_REG                0x04
#define IOADDR_ALT_ETH_10G_SCRATCH(base)       __IO_CALC_ADDRESS_NATIVE(base,ALT_ETH_10G_SCRATCH_REG)
#define IORD_ALT_ETH_10G_SCRATCH(base)         IORD_32DIRECT(base, ALT_ETH_10G_SCRATCH_REG)
#define IOWR_ALT_ETH_10G_SCRATCH(base,data)    IOWR_32DIRECT(base, ALT_ETH_10G_SCRATCH_REG,data)

/* Command register */
#define ALT_ETH_10G_CMD_REG                    0x08
#define IOADDR_ALT_ETH_10G_CMD(base)           __IO_CALC_ADDRESS_NATIVE(base, ALT_ETH_10G_CMD_REG)
#define IORD_ALT_ETH_10G_CMD(base)             IORD_32DIRECT(base, ALT_ETH_10G_CMD_REG)
#define IOWR_ALT_ETH_10G_CMD(base,data)        IOWR_32DIRECT(base, ALT_ETH_10G_COMMAND_CONFIG_REG,data)

#define ALT_ETH_10G_CMD_TX_ENA_OFST            (0)
#define ALT_ETH_10G_CMD_TX_ENA_MSK             (0x00000001)
#define ALT_ETH_10G_CMD_RX_ENA_OFST            (1)
#define ALT_ETH_10G_CMD_RX_ENA_MSK             (0x00000002)
#define ALT_ETH_10G_CMD_XON_GEN_OFST           (2)
#define ALT_ETH_10G_CMD_XON_GEN_MSK            (0x00000004)
```

图 10-1　Altera SOPC Builder 工具生成的头文件示例

10.3.3　RTL 工程师的接口

RTL 工程师需要将寄存器地址映射表连接到系统的其余部分。这部分工作包括为寄存器的每个比特位编写逻辑，以及为读/写周期创建地址译码器。在设计刚开始时就定义寄存器映射表，并且在整个设计周期内不断进行维护，这对 RTL 设计来说是一项极具挑战性的任务。因为在整个设计周期中的某些时侯，RTL 设计师有可能不得不修改寄存器地址映射表的部分内容。寄存器地址映射表的编码、技术说明书的编写、审核、以及与寄存器地址映射表的数据交流，整个过程是极容易出错的工作，很多 RTL 设计师不愿意进行此项工作。

幸运的是市面上有几个工具可以协助完成此项工作。FPGA 厂商提供的系统集成工具能自动生成软件头文件，在硬件系统设计工程师和软件工程师之间提供了一个自动化的接口。此外，这些自动生成的软件头文件还会负责生成地址译码

器的逻辑。

有些 EDA 工具可以提供更加高级的功能。这些工具能根据寄存器的描述为该寄存器地址映射表创建可综合的 RTL 代码，生成软件头文件和用来验证的头文件，还可以创建各种格式的用户使用说明书。

10.3.4　接口的验证

开发若干个测试平台对 RTL 级寄存器地址映射表的操作做全面的验证是良好的工程习惯。因此，验证工程师需要寄存器地址映射表的具体细节，而其格式要和其正在使用的验证语言一致。

作为验证周期的一部分，设计者应该依据设计规范规定的细节来验证软件对寄存器映射表的读/写。若将寄存器地址映射表的相关文档作为功能验证清单，该测试验证也可以在 FPGA 器件上进行。

10.3.5　文档

正如本章开头提到的那样，文档指的是设计团队的内部文件和提供给 IP 最终用户的文件。

每当寄存器地址映射表的 RTL 代码有改动时，设计师要负责更新相关的文档并与每个受到影响的小组一起评估本次改动。

描述寄存器地址映射表所使用的格式必须和所有设计者使用的命名规则保持一致。创建寄存器地址映射表的设计规范，用设计规范来指导文档的编写就可以做到这一点。

工业界已存在一个专为 IP 建立寄存器地址映射表的标准格式。这就是 IP-XACT 标准,该标准使用可以被市面上多个 EDA 工具读入的 XML 元数据(metadata)。然而,对于数据的写入,该标准还没有被所有 IP 供应商和 EDA 工具广泛采纳。

设计者也许应该考虑采用该标准而不是开发自己的格式，因此建议设计者在项目开始之前就认真评估该标准。

10.4　小结

寄存器地址映射接口是软件工程师和 RTL 工程师间的主要接口。在设计过程中不同部门都要使用该信息，为满足不同部门自身的需求，对该信息的格式要求有所不同。因此，必须对接口信息严加管理，任何有关接口信息的改动必须与需要该信息小组一起审核通过。由于实际中这项工作非常耗时，并且手动更新使用该信息的所有文件格式又极易出错，所以建议设计师购买专用于寄存器地址映射管理的 EDA 工具。

第11章　功 能 验 证

11.1　简介

每个设计小组都需要回答下面两个简单的问题：

1）设计功能是否正确？

2）验证过程是否完整？

设计师可能要花设计周期中60%以上的时间才能得到比较满意的回答。什么是正确的功能？验证到什么程度才算完？光把这两个问题说清楚就是一件艰难的任务。

以前，FPGA的设计规模很小，而且很多设计师并不关心设计重用的概念。那时候，FPGA设计师采取“开顺风船（blow and go）”的策略来验证FPGA设计。他们先编写RTL级设计模块，接着进行粗略的功能仿真，然后对FPGA器件编程，最后在系统中测试设计。如果发现问题，他们就会修改代码，并重复这个过程。然而对于规模大、功能复杂和质量高的系统而言，这种设计方法就行不通了。

FPGA器件的可编程特性确实给设计验证武器库里添加了一件功能强大的武器。然而，就可编程特性本身而言，它对创建可靠的和可重用的设计并没有什么用处。

许多出版物和EDA工具都致力于解决功能验证问题。

有很多不同的验证技术可用来验证设计是否已达到技术说明书所规定的要求。许多用于ASIC设计的验证技术也适用于FPGA设计的验证。如前所述，FPGA器件的可编程特性有助于以FPGA为实现目标的设计验证。本章将针对FPGA设计和（以FPGA为目标器件的）IP设计，阐述以功能验证效果好而闻名的验证技术。

11.2　功能验证面临的挑战

功能验证的高层次目标是为了验证指定的设计功能。这既适用于完整的设计，也适合于任何部件的设计。

功能验证必须覆盖设计的所有操作模式。这包括所有的边界情况。设计者最

不希望看到的是当设计成为产品后，由于所设计的系统进入到一种设计者事先没有考虑到或测试到的操作模式，从而导致产品出现严重的故障。

应用接口必须按设计预期运行，换言之，测试必须模仿设计和系统其他部分的互动。

在FPGA器件通过标准协议接口（例如PCI Express或Serial Rapid I/O）与系统其他部分接口时，有必要验证该接口模块是否与相应的标准兼容。

在设计参数化IP的场合，根据参数对该IP的所有结构变化做一次完整的测试是很有必要的。因为这样做可以增强IP使用者的信心，使他们相信该IP可以满足需求。

在IP为了可重用已经封装好，而且有用户接口时，必须验证用户接口能按预期运行，并且支持所有的操作系统。

最后，设计者需要审核IP模块的设计文档是否清晰完整，与IP核的行为是否一致。

这听起来像是需要做很多工作……而事实的确如此！

在规定的时间范围内利用可找到的资源，如何获得足够的验证覆盖度是设计师必须面对的挑战。

设计师如何确定覆盖测试已达到可以接受的程度呢？

这些问题的答案来自于验证计划。验证计划必须详细说明验证覆盖度目标和其他指标。因此，验证计划会影响项目的计划。

设计师在编写设计功能规范的同时，必须制订设计的验证计划。

在设计环节中，需要安排一个系统。该系统能够在设计的全过程包括验证周期期间，对照验证计划对设计的进展情况进行全面的监测。该系统必须能够管理测试生成的大量数据，并对照验证计划报告项目的进展情况。

11.3 有关验证的术语

1. 被测器部件（Device Under Test，DUT）：被测试的IP。

2. 断言（覆盖点）（Assertions（coverage points））：当设计正常运行时，由断言描述的设计工作状态为真。当设计运行不正常时，断言被激活。它有效地覆盖被测器部件（DUT）的所有状态。

3. VMM（Synopsys公司《SystemVerilog验证方法学手册》书名的英文缩写）：该书详细地介绍了用SystemVerilog对复杂设计进行全面验证的方法学。

4. 测试平台（Testbench）：测试平台是可在其上运行设计，并且设计的正确性能得到验证的运行测试环境。

5. 执行器（Transactors）：在测试平台环境中，执行器（transactor）是定义

事件发生顺序或任务执行顺序的模型。

6. 记分板（Scoreboards）：记分板是一种数据结构，该结构可保存预期的操作结果，以便与实际达到的结果相比较，

7. 寄存器抽象层（Register Abstraction Layer，RAL）：VMM RAL 自动为存储器映射的寄存器和存储器创建高层次的抽象层。在《SystemVerilog 验证方法学手册》中对寄存器抽象层有详细的描述。

8. 可执行的规范（Executable specification）：可执行的规范是描述设计功能、硬件和/或软件的高层次模型。它通常用高层次语言编写，例如 C、C ++ 、System C 或 SystemVerilog。

9. 回归测试（Regression Tests）：回归测试是每次设计修改之后，按既定规律，例如在每天晚上或者每个周末，结合具体应用进行的一系列测试。其目的是为了确保设计修改后没有引入新的错误。回归测试是一个用来证明完成的设计能按照说明书的要求运行的自动化测试环境。

10. 开放的验证方法学（OVM ，Open Verification Methodology）：由 Cadence 和 MentorGraphic 两家公司共同开发的一个标准 SystemVerilog 库和验证方法学。

11.4 RTL 仿真和门级仿真的对比

RTL 级仿真执行的是不考虑设计时序延迟的功能验证。RTL 仿真通常被用来验证设计的功能，而时序分析通常被用来验证设计中没有时序冲突。

门级仿真使用布局和布线后产生的时序网表。它包含用标准延迟格式（Standard Delay Format，SDF）表达的器件时序延迟。由于门级仿真包含时序信息，所以通过门级仿真，可以观察到设计在芯片上运行更为准确的细节。与 RTL 仿真相比，时序仿真需要花费相当长的时间。所以在实际工作中，很多设计师不愿意在诸如视频、图像处理等应用以及大规模设计中使用时序仿真。因此，建议只对关键的设计模块而不是完整的设计进行时序仿真，或者在系统硬件检查的调试过程中发现问题后，才进行时序仿真。

11.5 验证方法学

为了圆满地完成设计的验证工作，设计师必须采用多种验证技术。应该使用功能覆盖和代码覆盖相结合的技术，因为这些技术是相辅相成的。对于某些协议，设计师也应该进行硬件互操作性测试。

最后，不要忘了目标器件是可编程的。在设计的硬件实现中，有些故障很难找到，往往需要花费数天或数周的仿真时间才能解决。第 13 章将会更为详细地

描述系统在线调试技术。

具体的验证方法应该使用以下几个步骤。

11.6 克服复杂性

解决测试设计复杂性问题有三个主要的原则。

11.6.1 设计和测试的模块化

用单个测试就能对设计的所有功能做全面的验证是绝不可能的。因此，必须针对设计的不同方面编写不同的测试模块。设计和测试的模块化除了能提供一个更加周全的验证环境外，由于模块化的测试比较容易理解，因此测试移交时，其他人员接手也比较容易一些。

对大设计模块而言，设计师应采用第8章中所描述的功能验证方法。将设计分解成若干个较小的子设计，并在验证完整设计之前彻底地验证每一个子设计。

11.6.2 规划预期操作

创建测试是为了确认设计能在预期的或规定的操作模式下正常地运行。所完成的设计应该能在各种规定的条件下，使每一种操作模式都能正确地运行。因此，这些测试必须能对功能描述和设计说明书中列出的所有功能做逐项的验证。

令设计在各种边角情况下运行，确认在每种情况下，设计都能如预期的那样正常地运行。

作为功能测试的一部分，要确保设计师能对每个寄存器的比特位和每个端口上的每个信号进行操作。

当验证的设计由多个参数化模块组成，而且这些模块中的参数可由用户设置时，设计师必须运行所有可能的模式组合以测试连接模块间的互动。

每次操作后，必须验证系统是否已返回正确的状态。

11.6.3 应对意外状态的计划

由于存在没有被测试到的系统状态，系统运行有可能误入一个不能自恢复的状态，这是设计师最不希望看到的。因而设计师必须对那些可能出现的意外状态进行测试。这些意外状态因应用不同而不同，诸如向上溢出、向下溢出、CRC错误和运行中止等等。作为意外状态测试计划的一部分，设计师应该测试那些计划以外的状态，考察系统是否能从意外状态中恢复。意外状态不一定会出错，这是因为在实际系统中很可能不会出现这种意外状态。关键是所设计的系统是否能从极少发生的意外状态中恢复。

意外状态测试计划应该测试那些不会发生的状态，如：

1. 非法状态；
2. 违背设计前提的状态；
3. 违背协议的状态；
4. 在运行中模式变换的状态。

再强调一下，这里的关键因素是虽然设计的运行可能不正常，但它应该最终能恢复过来。

作为 IP 或设计模块功能验证的一部分，设计师应测试它和整个设计中其他模块间的互动，以保证接口操作按预期运行。

11.7　功能覆盖

如 9.6 节所述，为降低测试的复杂度，兼容性测试和临界测试虽然好，但其本身不足以全面测试所设计的系统。设计师不太可能预测和运行所有可能的条件，这增加了系统失效的风险。功能覆盖测试增强了设计师对所验证设计模块或系统的信心。首先确定有多少设计功能已经在验证环境中运行通过。再针对说明书中的其他特定功能逐一创建测试案例。其中的关键点在于设计师必须能够证明应该被检测到的功能都进行了测试。

模块和整个系统的测试计划应详细说明验证的覆盖度。这里的覆盖度是指功能覆盖目标（译者注：必须测试哪些设计功能，以及每个功能必须达到的指标）。

在制订功能覆盖目标计划时，设计师所面临的挑战是要确保该设计实现了说明书中所描述的功能；对于接口而言，则需要符合协议规范的标准。

设计师要按照制订的功能测试计划，逐项完成功能的测试，并令其与相应的最佳参考模型所描述的行为匹配。

对于可重用设计模块而言，设计师必须完成以下 4 个测试：

1. 被测器件的每个可用的特性和功能；
2. 各种配置的组合；
3. 可施加的不同类型的激励；
4. 被测试器件（DUT）的响应。

因为定义一张能验证 100% 设计功能的清单十分困难，所以功能覆盖度也有局限性。因此，在覆盖空间中找出覆盖的漏洞至关重要。

11.7.1　定向测试

定向测试要求为测试计划中的每个功能的测试编写一个测试模块。想要获得

可接受的功能测试覆盖度，人工编写测试模块的工作量非常巨大。而这些测试模块被重用的机会又非常小。

由于定向测试非常耗时，因此最好只用于典型功能的测试。

小模块的测试，建议采用定向测试技术，而大模块或系统的测试，则应采用随机约束测试技术。

11.7.2 随机动态仿真

在这种验证方法中，随机激励被用来增加功能测试的覆盖度。用高级验证语言能最好地发挥这个验证方法的作用。多年来为了增加测试覆盖度，开发了许多的语言和工具。SystemVerilog 是这个领域中涌现的佼佼者，已被 IEEE 协会批准并定为标准。它在多种验证语言中，得到了最广泛的工具支持。

建议设计师考虑采用 SystemVerilog 来验证系统设计。

11.7.3 受约束的随机测试

受约束的随机测试建立在随机动态仿真基础之上。最好在设计的早期运行随机仿真，以便发现大量的错误。随着设计接近完成时，为了充分覆盖测试空间，就需要对随机激励进行约束。

只需要运行一个受约束的随机测试就能覆盖测试计划中的多个测试项，这样可以缩短仿真时间。

这种方法也可以检测到测试计划中没有涉及到的问题/错误（见图 11-1）。

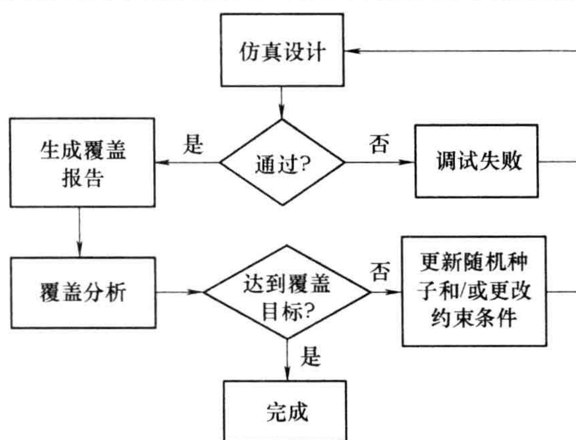


图 11-1 受约束的随机测试流程

11.7.4 SystemVerilog 用于设计和验证

实际上 SystemVerilog 将三种语言合为一体。

1. SystemVerilog 包含比 Verilog 和 VHDL 更为强大的可用于设计和综合的架构。

2. SystemVerilog 拥有更为先进的测试平台架构，以产生测试激励和覆盖。

3. SystemVerilog 支持断言结构，以捕捉设计师的意图。

SystemVerilog 本身支持由覆盖率驱动的受约束的随机验证。

市面上主要的 EDA 仿真器都拥有预先验证过的断言库选项。

目前，业界在 SystemVerilog 的验证方法上仍存在分歧，因而有 VMM 和 OVM 两个主要的验证方法库。把这两个库标准化，合并成为一个库，是发展趋势。

11.7.4.1　断言

断言被用来检查设计师所做的假设以及与设计有关的行为是否正确。在动态仿真过程中，如果设计满足或违反设计规范都可以触发断言。断言可用在模块级和系统级的测试中。

可重用模块中的断言也是可重用的，这是断言的另一个好处。

断言能及早发现问题，诸如 FIFO 的上溢/下溢错误。断言还能截获模块间的通信，例如存储器接口的行为。

11.7.5　通用测试平台方法

编写最简单的测试平台时，不必考虑验证代码的编写。只要工程师自己核对波形并判断仿真输出的波形是否与预期的一致即可。这种方法的缺点是设计师必须对设计有充分的理解，而且还要理解输出的波形才行；而别的工程师做仿真时，则必须花费很长的时间才能理解仿真输出的波形，因而故障漏检很可能发生。

这种观察波形的测试方法最适合用在不准备重用的简单模块的设计中。

设计师编写“测试装置和接线”代码，产生激励信号，施加到实例引用的设计模块上（见图 11-2）。

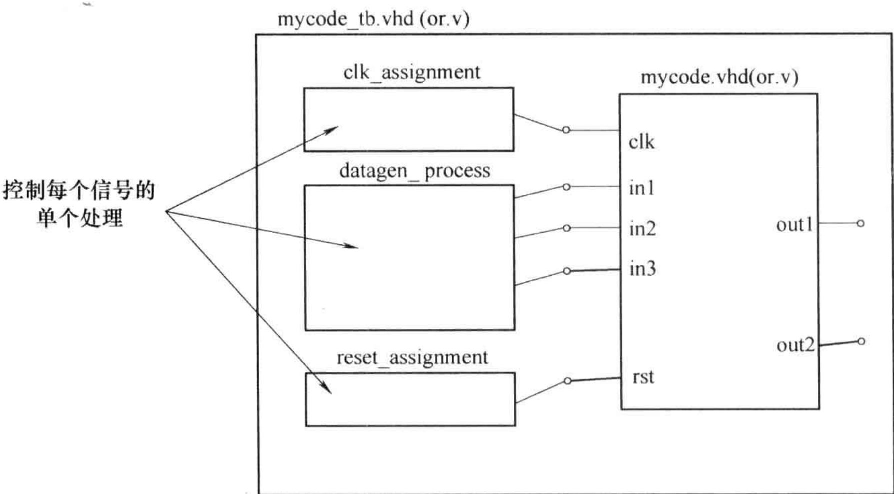


图 11-2　需要人工检查波形的简单测试平台

11.7.6 自验证测试平台

编写自验证的测试平台比较困难一些。必须充分理解待测试模块，才有可能写出“期望的输出”。由于“期望的输出”中的错误很难找到，使用这种方法必须事先做相当多的工作。然而一旦自验证的测试平台设置后，设计师就可以进行测试，很快就能得到通过或失败的结论。

测试可重用的设计模块应该使用自验证的测试平台。

在编写自检验测试平台时，为了能对输出进行监测，设计师必须在已有的处理过程中添加一些功能，例如添加一个“比较进程”或类似的进程，用以将仿真得到的结果与预期的结果进行比较（见图 11-3）。

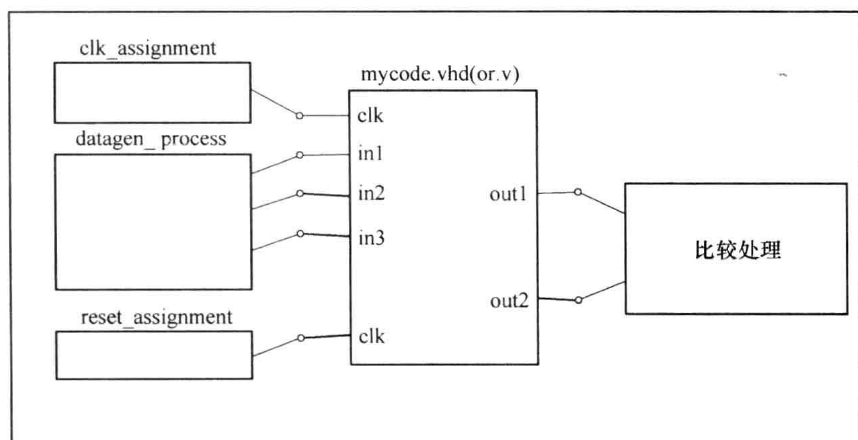


图 11-3 自检验测试平台图例

这类测试平台可以产生顺序的或并发的激励，也可以用期望的输出向量产生这些激励。

通常发出的信令结构非常复杂，如果不用保留在“时间片”中的向量，很难为其建立模型。在自验证的测试平台上，利用内部数组或外部文件，可以为复杂信号结构建立模型，从而产生极其复杂的信令激励，自动地对系统进行测试。

在测试平台中，使用包含激励信号和预期输出信号向量的数组时，可以不需要进行类型转换。这虽缩短了仿真时间，但编写测试激励比较困难，可能要创建非常大的文件。

当使用包含激励和期望结果向量的外部文件时，可能需要进行数据类型的转换。这有可能导致仿真变慢，但编写测试平台则相对比较容易一些（见图 11-4）。

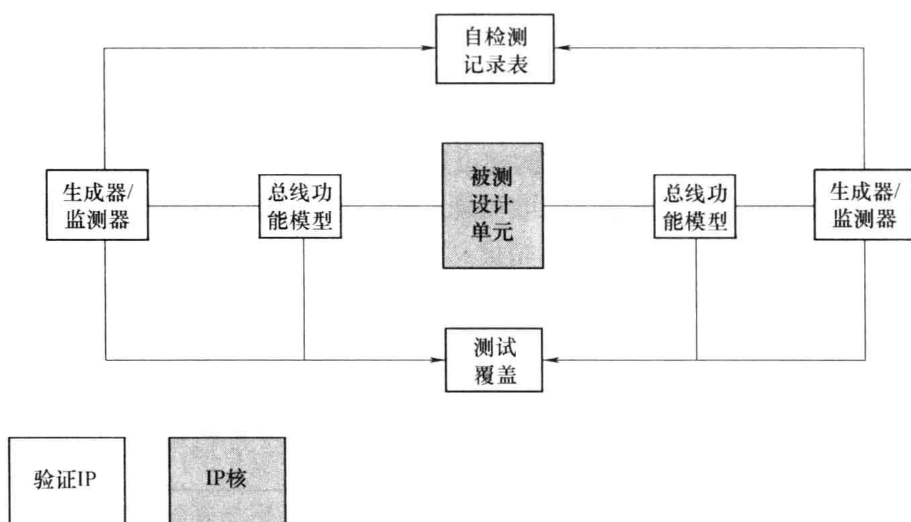


图 11-4　验证系统的架构

11.7.7　形式化等价性验证

形式化等价性验证比较设计流中或不同网表中不同点之间的逻辑等价性。它使用数学方法来比较同一个设计的两种版本在逻辑上的等价性，而不是使用测试向量进行仿真。

形式化等价性验证通常被用于 RTL 代码与综合后门级网表之间的比较，以此来保证综合优化没有引入任何错误。它也可用于 RTL 代码或综合后门级网表与布局布线后网表之间的比较，以确保布局和布线优化没有改变设计的功能。

虽然形式化等价性验证可以确定两个网表是否在功能上具有一致性，但它并不能保证其功能的正确性。如果 RTL 设计实现的功能已经不正确了，而与之相比较的网表具有相同的功能，形式化等价性验证也会报告“成功”。因此，形式化等价性验证通常用于已通过功能验证的 RTL 代码和门级网表间的比较。

形式化等价性验证工具能支持的设计规模往往有限，因此它们大多用于设计模块的验证，而不用在完整设计的验证中。

但对于 FPGA 而言，这个技术使用起来特别困难。FPGA 综合优化过程需要对寄存器进行多种优化，如寄存器合并，寄存器复制和寄存器重定时。前两种优化可能会导致伪失效报告。对设计仔细研究后，可以排除这些伪错误，但却非常耗时。第三种优化（即寄存器重定时）经常会造成逻辑混乱。大多数形式化等价性验证工具因为无法对付由 FPGA 逻辑综合或物理综合进行的寄存器重定时所造成的逻辑混乱，因此在 FPGA 设计流程中几乎不使用形式化等价性验证。

11.8 代码覆盖度

代码覆盖度反映的是HDL代码运行的彻底程度。

代码覆盖度能提供有多少行代码已被执行的信息，它提供了定量计量测试质量的手段，因此有助于指导今后测试工作的方向。

代码覆盖测试是有局限性的，因为它并不关注事件发生的先后顺序，也不检测设计模块间的任何交互。代码覆盖测试只关注设计中有些什么，并不关心没有实现的功能。总而言之，代码覆盖测试不关注设计的具体功能。

代码覆盖测试能发现在随机测试下没有遍及的边角情况。为了做到这一点，用户必须编写定向测试案例，对没有被覆盖的区域进行测试。

11.9 质量评价（QA）测试

11.9.1 功能回归测试

功能回归测试的目的是企图提供一个自动化的测试环境以证明设计能按指定的要求正确地运行。

回归测试必须确保旧错误不再重现。在调试中找到并改正错误之后，应立即编写测试程序以检测该错误是否已被纠正，这是一个良好的工作习惯。以后对设计做任何改动时，都应运行这个测试程序以保证新改动没有引入新错误。回归测试将自动执行这个测试过程。该测试应该被归并到整个设计的测试套件中，其中所有的回归测试案例都将在这个测试环境中自动地执行。

自动化质量评价（QA）回归测试通常通过脚本执行对IP或设计的自动仿真。在仿真中把编译和运行的结果与一个已知的合格规范进行比较。这个测试是自检测性的，会产生验证日志，报告意外情况。请注意意外情况这个术语的使用。在测试中发现问题属于意外情况，除非通过分析，确认测试所发现的问题确实是由设计错误所引起的，才算发现设计错误。通常意外情况是由测试环境问题引起的，而不是由设计错误引起的。如果确实是测试环境的问题，就应该在解决该测试环境问题后重新测试，观察测试是否通过。回归测试环境必须能编译测试的统计结果，并报告设计的健康状态。其中包括单个设计模块的报告，也包括集成了所有设计模块的最终系统设计的报告。

11.9.2 可重用IP的图形界面（GUI）测试

带图形界面的IP虽然简单易用，但必须通过严格的测试才能确保用户能正

确地使用。其他用户第一次接触你的 IP 可能就是这个图形界面。你总希望自己设计的 IP 能给用户留下好印象，千万不要出现由于图形界面中的问题使得你的 IP 不受欢迎的情况。

市面上有测试程序可以用来对图形界面进行回归测试，然而手动测试仍是最有用的测试手段。

测试的目的是：

1. 确保参数化图形界面的功能与预期的一致；
2. 在正常使用情况下，确认 IP 的性能指标；
3. 在错误的使用条件下，检查 IP 的行为。

该测试需要由人按照测试清单对图形界面的运行情况进行逐项细致的测试。测试人员点击按钮、载入文档、核查预期结果并编写错误报告。

这种测试方法相当耗费时间和人力，但可以保证用户在使用图形化接口时有良好的体验。

11.10　硬件互操作性测试

在设计和标准协议接口的地方需要进行硬件的互操作性测试。可按工业标准 ASSP 在实验室对硬件的互操作性进测试，也可以在互操作性（industry plug-fests）和测试实验室对其进行测试。

11.11　软/硬件协同验证

市场上可以购买到软/硬件协同仿真的工具。这种工具能以很高的效率在硬件模型运行‘c’代码。由于‘c’代码在仿真软件上的运行速度非常慢，比它在硅片上实际运行速度慢几个数量级。因而为了加快仿真速度，通常在开发板或终端系统的 FPGA 器件上运行‘c’代码，以避开用仿真软件进行测试，这是 FPGA 设计中常用的技术。

11.11.1　加快投片的准备

利用 FPGA，我们能快速地得到电路板级的初步设计。在电路板级的系统测试中能发现用 RTL 仿真无法检测到的错误。想要验证设计是否正确，必须将硬件检查与仿真结合起来。在 FPGA 设计的初期，通过仿真可发现设计中存在的绝大部分问题，所以此时仿真最有价值，而在调试接口和驱动程序时，硬件检验非常有用。

11.12 功能验证清单

1. 编写测试计划，其中应包括能证明设计正确无误的重要测试案例的每个细节。

2. 编写功能覆盖度测试说明书，其中应该定义功能覆盖的测试范围。

3. 搭建系统测试平台。

4. 编写功能测试模块，并进行仿真，以期获得功能测试的覆盖度。

5. 进行代码覆盖度测试，只有在 RTL 代码稳定之后才能进行这项测试。

6. 实现彻底的功能覆盖测试，如果设计模块的功能测试覆盖率达到 100%，就应该扩展到系统级的功能覆盖测试。

7. 对 IP 进行图形界面测试。

8. 对具有标准接口协议的 IP，必须完成硬件互操作性测试。

9. 执行系统调试。这次调试需要将软件下载到目标硬件上，执行软/硬件协同仿真，以验证软件与硬件能很好地配合，正确无误地运行。

第 12 章　时 序 收 敛

12.1　时序收敛的难点

FPGA 设计流程中，时序收敛是设计师最易产生挫折感的阶段。时序收敛有可能耗尽 EDA 工作站的机时。即便如此，设计的性能如仍不能达标，设计师就不得不选用速度等级更高的 FPGA 器件，从而造成开发经费超标。

本书中的大部分章节都是围绕着如何在设计中克服困难，达到时序收敛的目标而展开。

在进入到下一个阶段的讨论之前，本章先介绍达到时序收敛目标的设计方法。

为什么说在 FPGA 设计中达到时序收敛目标是一项艰巨的任务呢?

在过去十年间里，FPGA 器件的密度和面向 FPGA 器件的设计规模都有巨大的增长。FPGA 器件的逻辑密度增长了约 30 倍，嵌入存储器的数量增加了约 70 倍。然而在同时期内，工作站 CPU 的速度却只增加了 14 倍。以上这些因素就造成了高密度 FPGA 设计的编译时间过长。

除此之外，设计的时钟速度和接口速度都有显著地提升。目前 FPGA 器件中收发器的速度超过了 11G，DDR3 内存接口的运行速度超过了 533MHz。

这些类型的应用要求更复杂的时序约束，例如源同步接口和跨时钟传输。

最新的 FPGA 器件工艺尺寸小，为达到时序收敛的目标，要求时序分析在两个或多个时序的边角条件下进行。对这些工艺尺寸小的 FPGA，在延迟中占主导的通常是互连线延迟，而不是逻辑单元延迟。在避免布线拥塞的同时还要避免冗长的互连延迟，这就给 FPGA 设计的布局带来了困难。

添加专用的硬件模块，如嵌入式存储器和 DSP 模块，带来的好处是增强了功能，但如何安置与这些模块接口的相关逻辑也是个难题。

值得庆幸的是，FPGA 厂商所提供的软件中包含了很多个功能项来解决这些困难。在多数情况下，默认设置就能满足设计的性能指标。对那些按默认设置不能满足指标的设计，设计师可以借助众多的分析工具和功能项使设计达到要求。

12.2 时序分配和时序分析的重要性

在提到时序收敛时，时序分析是设计师必须理解的唯一的最重要问题，也是最不容易理解的地方。

本章这一节将说明时序分析的重要性，并且给出时序分析的基本背景。深入的时序分析本身就可以写一部书。为了更好地理解时序分析，建议设计师参加FPGA厂商的培训，并且从它们的网站下载各种操作说明书。

时序分配在FPGA设计中有如下两个作用：

1. 指导综合工具进行布局布线工作。时序分配对布局布线的影响将在12.3.4.1节“认识布局布线工具”中详细介绍。时序分配告诉综合器应该把优化的重点放在哪里，并命令布局布线工具，优先考虑那几条路径的布局和布线。

2. 用于时序分析。时序分析并不保证RTL设计功能的正确性，只保证设计没有时序冲突。静态时序分析通过计算得到设计时序，并没有进行仿真。

12.2.1 时序分析的背景

回顾过去，FPGA设计流程中的时序分析曾经比较简单。当时由于终极应用相当简单，时钟域的个数有限，而从厂商那里得到的时序模型又设置了超大的保护带，因而设计师只需在一个时序边角条件上分析设计就足够了。每个FPGA厂商主要针对时钟频率，创建了它们自己的时序分配语言。这样就有效地庇护了设计师，使得他们无需知道时序分析的复杂性。

对于当今这类面向FPGA器件的设计，设计师在时序分析上面临的困难和ASIC设计师多年来面临的困难相同。如今典型的FPGA设计通常使用多个时钟域，时钟域之间又有复杂的关系，因而需要特别注意接口时序，而不是只是单纯地找到最高的时钟频率。对于最新的65nm和45nm工艺，为了保证FPGA设计的正常运行，必须在多个时序的边角条件下进行时序分析。由于最初的时序分析语言不是用来约束这类FPGA设计的时序的，这就要求FPGA设计师必须学习ASIC的时序分析技术。

值得庆幸的是，FPGA厂商和EDA工具行业正在标准化来自Synopsys的SDC（Synopsys Design Constraints）时序约束语言。

12.2.2 时序分析基础

本节除了简单描述构建时序分析的初级时序约束之外，还要解释在时序分析中使用到的通用术语。

12.2.2.1 静态时序分析

静态时序分析估算设计中时序路径的时延，并对照设置的时序约束，报告映射后实现电路的时序。基于 FPGA 硅片的时序特性，静态时序分析能确定设计是否能正常运行。时序分析与输入的功能无关，它依据时序要求，遍及分析设计中所有输入和器件中每条路径可能的组合，从定电路的延迟。

与门级仿真和电路板测试比较，静态时域分析能更快更容易地发现与时序有关的错误。

12.2.2.2 设计约束（SDC）

SDC 是 Synopsys Design Constraints 的缩写。它是针对时序约束的行业标准语言，已为大多数 FPGA 厂商和支持 FPGA 器件的 EDA 工具所采纳。

12.2.2.3 时钟

时钟被用于触发寄存器到寄存器的同步传输，引导综合和布局布线工具实现优化算法，以达到最好的设计效果。

在任何设计的约束（SDC）文件中，应该首先确定时钟约束。时钟约束之所以重要是由于很多约束都需要有参照时钟才能执行；因此必须最先定义时钟约束。

12.2.2.4 发送沿

发送沿是从寄存器（时序元件）发出数据时使用的有效时钟沿。例如寄存器作为数据发送端时就要使用该触发沿。

12.2.2.5 锁存沿

锁存沿是在时序元件的数据输入端捕获数据时使用的有效时钟沿。例如寄存器作为数据接收端时就要使用锁存沿。

发送沿和锁存沿的关系，如图 12-1 所示。

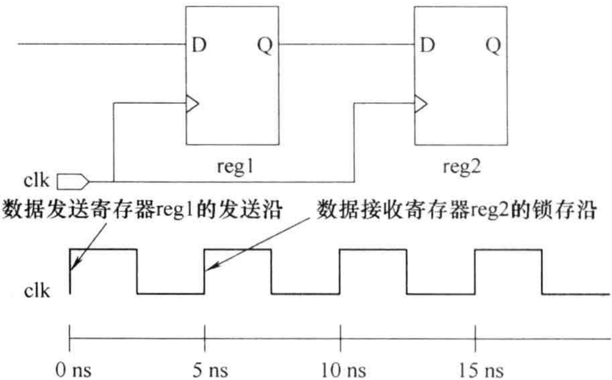


图 12-1 发送沿和锁存沿关系示意图

12.2.2.6　保持时间（t_h）

为了让某个数据存入寄存器，在触发该寄存器的有效时钟沿到达其时钟引脚后，该数据必须继续在寄存器的输入（或者使能输入）端口保持一段时间不发生变化，才能确保该数据准确地存入寄存器。数据在寄存器输入端必须继续保持不变的最短时间就是 t_h（保持时间）。

寄存器（时序元件）的有效触发时钟沿到来之后，输入信号变化太快会造成保持时间不够，从而导致寄存器（时序元件）输出时序混乱。

12.2.2.7　建立时间（t_{su}）

为了让某个数据存入寄存器，在触发该寄存器的有效时钟沿到达其时钟引脚之前，该数据必须在寄存器的输入（或者使能输入）端口已保持了一段时间不发生变化，才能确保该数据准确地存入寄存器。数据在寄存器输入端必须已保持不变的最短时间就是 t_{su}（建立时间）。

建立时间和保持时间的关系，如图 12-2 所示。

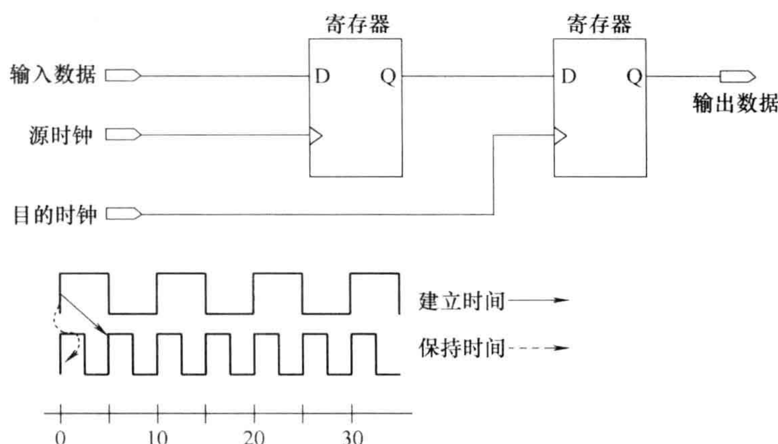


图 12-2　建立时间和保持时间示意图

当信号到达寄存器（时序元件）的输入端过晚，错过了它应该传到下一级寄存器的时间（这里时间是指下一级触发器的触发时钟沿到达时刻），就会出现建立时间不够的时序冲突。建立时间不够会导致寄存器（时序元件）时序混乱。

12.2.2.8　到达时间

到达时间可分为数据到达时间和时钟到达时间。

数据到达时间指的是数据从源时钟到目的寄存器（数据输入端）的延迟。

时钟到达时间指的是从目的时钟节点到的目的寄存器（时钟输入端）的延迟。

数据到达时间和时钟到达时间的关系，如图 12-3 所示。

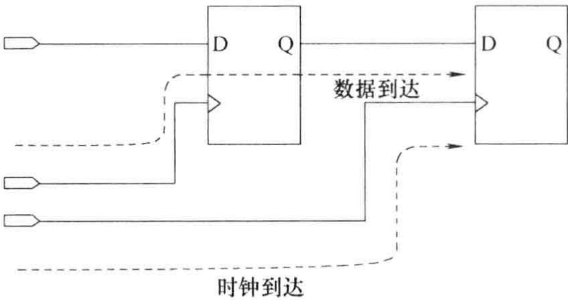


图 12-3　时钟到达和数据到达示意图

12. 2. 2. 9　时序要求

时序要求是指在不延长设定的时钟周期的前提下，对信号最晚必须到达时间的要求。

12. 2. 2. 10　时间裕量

时间裕量是指满足或不满足时序要求时所留有的裕度。它是要求（信号）到达的时间和（信号）实际到达时间的差值。正裕量是满足时序要求时，还留有的余地。而负裕量则是在不满足时序要求时，与时序要求间的差距。

12. 2. 2. 11　时序异常

原本不需要时序异常这个约束，然而为了更好地描述设计如何运行，添加了时序异常这个约束。时序异常可调整如何对设计进行时序分析。多周期路径和伪路径是时序异常的具体实例。

12. 2. 2. 12　多周期路径

信号更新的时间超过一个时钟周期就是多周期路径。这些路径需要由模块的设计师来确定，确定哪几条路径为多周期路径，需要对设计的功能有透彻地了解。

分配多周期路径，就是允许设计师指定数值，锁定到目的寄存器之前需要等待的时钟周期个数。因此，目的寄存器就能在某个准确的时钟沿到来的时刻锁定该数值。

图 12-4 详细说明了所需要时钟个数为 2 时，多周期路径下的数值寄存。数值锁存触发沿比单周期路径延迟了一个目的时钟周期。

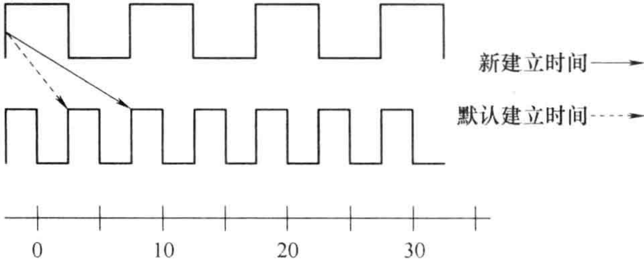


图 12-4　多周期路径

12.2.2.13 伪路径

伪路径分配，用来明确指出哪些路径不需要进行时序分析。例如测试逻辑或所有与电路运行无关的路径都不需要进行时序分析，因此设计者必须把这些路径明确地指定为伪路径。跨时钟域的路径通常被指定为伪路径。

12.2.2.14 源同步

源时钟同步是用于描述时钟和数据同源的技术。在源同步接口中，时钟源与数据源使用同一设备。

12.2.2.15 上升/下降时间

上升时间是指信号值从低变到高所需的时间。低值通常是信号值的10%，而高值为信号值的90%。下降时间是指信号值从高变到低所需的时间。

12.2.2.16 输入延迟

输入延迟（建立_输入_延迟）规定了在指定的输入端口相对于时钟所要求的数据到达时间。输入延迟规定中的相对于时钟是指相对于该时钟的上升沿或者下降沿。(见图12-5)。

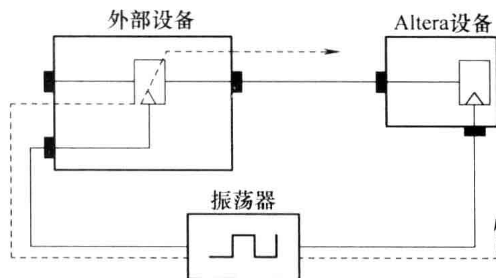


图12-5 输入延迟

12.2.2.17 输出延迟

输出延迟（建立_输出_延迟）规定了在指定的输出端口相对于时钟所要求的数据到达时间。输出延迟规定中的相对于时钟是指相对于该时钟的上升沿或下降沿（见图12-6）。

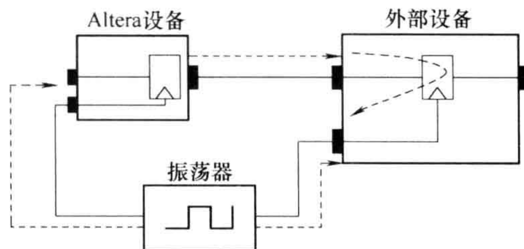


图12-6 输出延迟

12.2.2.18 运行条件

运行条件由设计时序分析期间所使用的电压设置与温度设置组成。这些设置

值会影响时序分析工具所使用时序模型的延迟。

12.2.2.19　多边角条件分析

在对设计进行静态时序分析时，采用多边角分析可以了解 FPGA 设计在不同运行条件下的时序情况。这种验证通常用设计的最慢边角模型和最快边角模型分两次进行。

在完成时序设计的签收之前，设计师必须进行多个边角条件的时序分析，检查是否达到时序需求。多年以前，FPGA 厂商只提供一个代表最差运行条件的时序模型。该模型内部有足够宽的时序保护带，因而设计师只需要用一个时序模型就可以完成时序分析，还能保证设计的时序符合要求。随着 FPGA 器件工艺尺寸缩小到 65nm、40nm 以下，这种做法就不再符合实际了。设计师必须在最好和最坏的条件下分别对设计做时序分析，待时序达到需求后，才能签收。也就是设计师必须分别针对最好和最坏两种运行条件对设计时序进行优化。

12.2.2.20　慢边角条件模型

慢边角条件时序模型表示在最差的运行条件下，任何单条路径可能的最慢性能。该模型表示在最高运行温度和最低电压（VCCMIN）下器件的最慢行为。通常慢时序模型用于检查建立时间是否能满足需求。

12.2.2.21　快边角条件模型

快边角条件时序模型表示在最好的运行条件下，任何单条路径可能的最快性能。

该模型表示在最低运行温度和最高电压（VCCMAX）下器件的最快行为。通常快时序模型用于检查保持时间是否能满足需求。

在最好的运行条件下，短路径的是否满足时序要求可以用这种分析来验证。

12.2.2.22　时钟的不确定性

时钟的不确定性通常是指时钟或时钟到时钟传输的偏斜。时钟的不确定性可分为建立时钟的不确定性和保持时钟的不确定性，还可分为时钟正跳变沿的不确定性和时钟负跳变沿的不确定性（见图 12-7）。

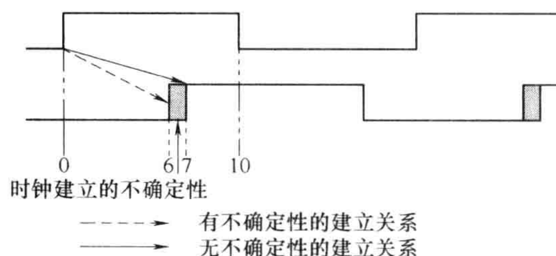


图 12-7　时钟的不确定性

12.2.2.23　时钟延迟

有两种类型的时钟延迟：网络延迟和源延迟。网络延迟是指时钟网络上时钟

和寄存器时钟引脚之间的延迟。

源延迟是时钟和时钟源（如系统时钟或产生时钟的基时钟）之间的时钟网络延迟。

当时钟输入端口被用作反馈时钟时，为清晰表示从时钟输出端口到时钟输入端口的电路板级延迟，源延迟可以被分配给所产生的时钟。

12.3 实现时序收敛目标的方法学

这一节将描述在 FPGA 设计中总能让设计师达到时序收敛目标的设计方法学。

12.3.1 指定 FPGA 器件系列

12.3.1.1 选择速度等级

为了能很快地实现时序收敛的目标，建议在设计之初就选用速度等级最高的 FPGA 器件来实现设计。这样就能更快地转入电路板功能验证阶段，并能更快地开始软件的开发。

设计师可以在验证周期中或功能验证完成后，再转用低速 FPGA 器件，对设计进行优化。

12.3.1.2 设置 I/O

设计师对驱动强度和 I/O 标准的选择会影响引脚的时序。它们也会影响所选器件的功耗和信号的完整性。

以下是可以用来改善 I/O 时序的技术，按优先顺序排列如下：

1. 确保给 I/O 引脚设置恰当的时序约束。

2. 检查报告文件，确定是否使用了 I/O 寄存器。如果没有使用 I/O 寄存器，就需要查看 RTL 代码，并重新编写 RTL 代码令输出寄存器驱动引脚，且令引脚驱动输入寄存器。为了满足 I/O 的时序要求，布局布线软件通常会自动使用 I/O 寄存器。若确实没有使用 I/O 寄存器，则设计师可以通过 FPGA 设计软件的设置项，强行令布局布线软件使用 I/O 寄存器。

3. 关注 I/O 单元的延迟链设置。对于引脚之间的连接，无论输入或输出引脚都要使用最短延迟。大多数 FPGA 器件的 I/O 单元具有可编程的延迟选项，用以最小化 t_{su} 和 t_{co}（建立和保持总时间）时间。这些延迟通常由 FPGA 设计软件根据 I/O 时序设置自动设定。若自动设定延迟没有起作用，则设计师可以通过软件中的设置项，手动设定。

4. 移动锁相环时钟沿。来满足 I/O 的时序要求如果锁相环为驱动 I/O 引脚的寄存器或 I/O 引脚的输入寄存器提供时钟，就可以对锁相环的输出进行相移以

改变 I/O 的时序。后移锁相环时钟以缩短建立时间 t_{su} 为代价从而提供更富裕的 t_{co}（建立和保持总时间）。前移锁相环锁相环时钟以缩短 t_{co} 和保持时间 t_{hold} 为代价，从而提供更富裕的建立时间 t_{su}。

12. 3. 2　设计规划

第 8 章所提到的那样，对时序收敛进行预先规划非常重要。预先规划会有助于在问题出现之前发现问题，并避免延误设计周期。

所有的 RTL 代码都编写完后，才去编译顶层设计是时序收敛中常见的误区之一。为了尽早发现设计集成和资源使用中的问题，设计师应该每完成一个主要的低层次模块 RTL 编码，就即刻对顶层设计进行一次编译。

为了实现顶层设计的时序收敛目标，设计师需要在设计说明书阶段，即定义如何划分功能模块时就要开始考虑时序收敛问题。时序收敛计划包含单个模块的时序要求、模块间的时序要求，以及对所有与专用硬件模块或器件引脚接口的模块布局的约束。在编译顶层 RTL 代码时需要遵循这些规定。关于 RTL 设计划分的更多详细信息见 8. 5. 2. 3 节。

这里还是要建议设计师使用增量设计方法。在实际设计中，通过如 8. 5. 2. 3 所述的合理设计划分，实际上已经为使用增量编译法进行设计做好了准备。对 FPGA 设计而言，这种方法的优势在于把它应用到基于团队的设计方法中十分容易。这样，很多位工程师可以一起做同一个 FPGA 设计，轻松地达到时序收敛的目标。这种设计方法可以使设计修改（即修改工程指令）的次数减至最少，从而减少对设计造成的影响。

主流 FPGA 厂商和 EDA 供应商所提供的 FPGA 设计软件都支持增量设计方法学。

12. 3. 2. 1　增量编译

正如之前所提到的那样，源自 FPGA 厂商的增量编译功能可以大大地缩短编译时间，但这不是该方法的唯一优势。增量编译方法还能缩短实现时序收敛目标的时间。使增量编译产生效益的关键因素是良好的设计规划。

那么，增量编译是如何工作的呢?

增量编译不编译设计中无变动的模块，只编译设计中有改动的那部分模块。由于需要重新编译的逻辑减少了，编译的工作量也就减少了，因而增量编译的最大好处是缩短编译时间。第二个好处是一旦设计中时序临界的模块满足了时序需求，设计师就可以对它进行锁定，这些模块的性能就会在整个设计中得到保持。而通常被忽略的第三个好处是设计师在加入调试逻辑进行调试时，使用增量编译不会影响设计的性能。这一点将在第十三章中更详细地讨论。

综上所述，设计师应该采用增量设计的方法。

然而设计师也应意识到增量设计法在设计应用中的限制，以避免一些易犯的错误。这些限制如下：

1. 它需要事先规划设计分区，如8.5.2.3节所述。这对设计模块如何接口就有了限制。

2. 它妨碍了跨模块的优化。该限制可以通过以下方法来解决：保持设计模块中的关键路径；寄存设计模块的端口信号；不要在下一层设计模块间插入组合逻辑。

3. 它降低了应能达到的器件利用率。有些FPGA设计软件在对完整设计进行面积优化时确实比较有效。例如，为节省面积，面积优化工具可把表示组合逻辑的查找表（LUT）与该组合逻辑无关的寄存器布在同一个逻辑单元中。若设计师想要在设计中充分利用每个逻辑单元中的每个基本元件，但由于FPGA器件中可用布线资源分配的缘故，设计师很可能无法实现时序收敛的目标。为了实现设计的时序收敛和高性能目标，有时必须牺牲FPGA器件的利用率，决定设计选用哪一种规格的FPGA，应在设计说明书中明确地加以说明。使用增量设计方法，可使大多数设计达到85%以上的逻辑利用率，同时实现时序收敛的目标。

自顶向下的设计流程

在自顶向下的设计流程中，整个设计在一个工程中进行编译，并针对整个设计进行时序收敛。当设计中不同模块的RTL代码完成后，它们被添加到顶层设计并和设计的其余部分一起编译。使用这种技术的优势之一是对分区之间的路径提供了良好的能见度。时序收敛是针对整个设计进行的。一旦设计师对所设计模块的结果满意，这个模块就被锁定，再也不需要被重新编译，从而在缩短编译时间的同时锁定了性能。

自底向上的设计流程

在自底向上的设计流程中，各模块在单独的工程中进行编译，一旦模块达时序收敛目标，就立即被锁定。低层分区在最后集成时移植到顶层工程中。这样设计不需要被重新编译，但需要融合所有模块的布局布线网表，然后通过布线操作连接所有的模块（见图12-8）。

自底向上的设计流程在不同团队成员间存在简单的设计分区时有帮助，但缺点是完全隔离了低层模块。它要求事先对芯片的资源分配花更多的功夫。为适应每个模块在独立的工程中进行编译，还需要进行详细的版图规划。它也使得整个项目的时序约束更为复杂，因为时序约束需要由顶层工程传递到低层工程中。任何需要在低层工程中添加的时序约束也必须移植到顶层工程中（见图12-9）。

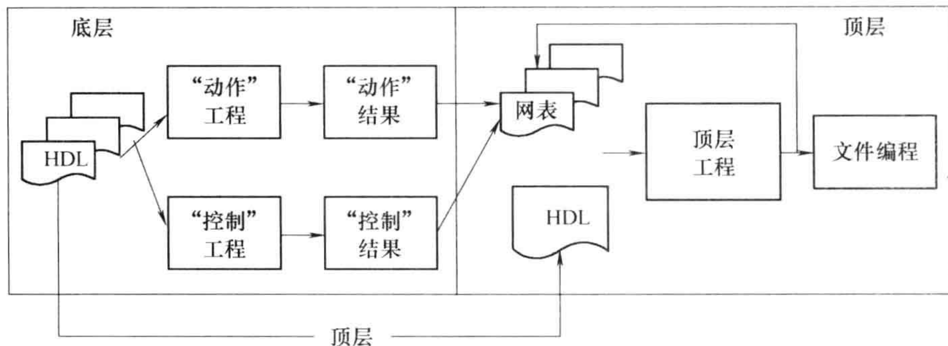


图 12-8　自底向上的设计流程

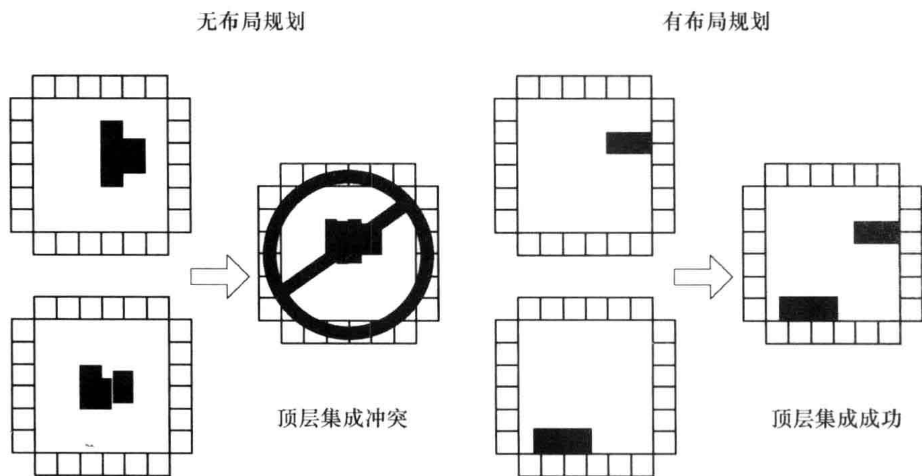


图 12-9　顶层设计中的模块集成

12.3.2.2　增量编译使用场合介绍

本节中，我们将关注几个使用增量编译能显著缩短时序收敛周期的场合。

图 12-10 展示了一个能进行增量编译的设计分区。

为了进行增量编译，该设计规划将设计划分为三个主要的层次：“动作”层、“控制”层和“顶层”。顶层是设计的最高层次，含“动作”模块、“控制”模块和其他层次。“动作”模块进一步地分层，含有两个其他的设计层次。“控制”模块是顶层设计模块之一“解码器”单元的一个子集。这个设计已经被编译并达到了性能要求。

场合 1：参数微调

在本场合中，由于设计说明书中一个小变动，系统需要做一些微调，会影响顶层文件中的存储模块，当“动作”模块和“控制”模块的 RTL 代码不再改变

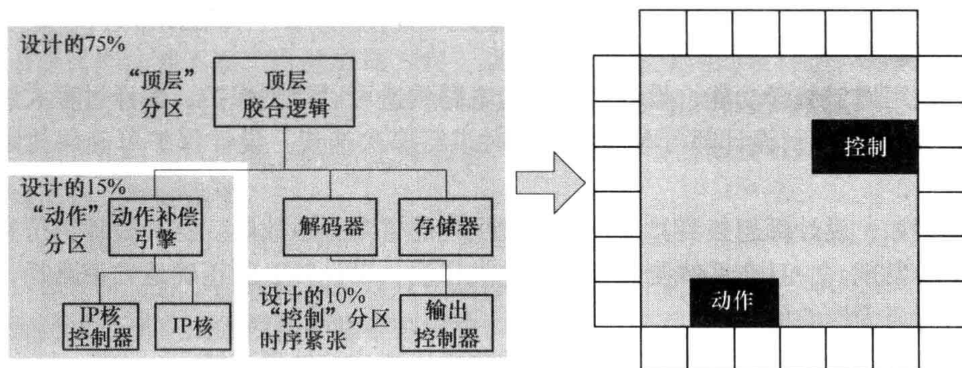


图 12-10 为进行增量编译的设计分区示例

时，设计师便可以将它们的布局布线锁定。需要改变的只剩下“存储”模块，此时只需重新编译顶层模块即可。由于“动作”模块和“控制”模块不需要再次编译，它们的性能就不会改变。这样只有 75% 的设计需要重新编译，也没有触及布局布线有困难的时序临界的模块，因而编译时间大大缩短了。

通常可在 6 个小时内完成编译的设计，重新进行一次完整的编译意味着设计师只能在一个正常的工作日内完成一次设计迭代。也就是一次迭代只能完成一次设计更改。

而使用增量编译的方法，编译时间可以降到不足 4 个小时，在一天内就可能完成两次设计迭代。若这些部分设计的时序不临界，则设计师可以使用 12. 3. 3 节介绍的早期时序估算中所描述的快速编译选项，那么在一天内就可完成更多次设计迭代。

场合 2：故障修复

在本场合中，若设计已完成，设计师正在实验室中对设计进行最后阶段的系统在线测试。系统正在高速地运行，设计师突然发现一个功能故障，他必须尽快找出故障的原因，并及时修复。

设计师利用 FPGA 厂商提供的一些可用的调试选项，可以保护整个设计的布局布线，而无需再做一次完整的编译。

设计师可将设计中的内部信号快速地连接到未使用的引脚上，而不会干扰设计的布局布线。

设计师可添加 FPGA 厂商提供的嵌入式逻辑分析仪而无须对“顶层”模块、“动作”模块和“控制”模块重新编译。如果设计师想隔离错误，可以完善嵌入式逻辑分析仪的触发条件并快速地创建一个新的编程文件。

一次完整的再编译需要 6 个小时并且会改变设计的实现。不使用增量编译方法，添加嵌入式逻辑分析仪或者改变嵌入式逻辑分析仪可能会导致原错误消失；

设计师就会疑惑：该设计的功能到底是正确呢？还是不正确呢？这个问题会在产品中再现吗？

使用增量编译功能，设计所实现的电路构造可以得到保护，编译过程大约需要花费 45 分钟；设计师可在设计调试时进行多次迭代。设计保护可确保故障重现。

例如，设计师想找到异步信号经常出现的冒险竞争故障。这种故障用仿真工具很难发现。一旦在系统测试中发现这类错误，设计师必须正确地约束路径，并重新编译受影响的模块。

建议设计师只对高速运行时出现的故障采用这种处理方法。

场合 3：时序收敛

在本场合下，为了提高设计的整体性能，有必要考虑提高时序收敛的目标。若从第三方得到一个新版本的 IP 核，则可能需要考虑时序收敛。在图 12-10 所示的实例中，若必须使用一个新版本的“动作”核，则设计说明书也得做相应的改变，模块的性能必须从 120MHz 提高到 150MHz。

设计师在编译某设计时，其“动作”核的时序收敛目标无法实现。由于“动作”核是来自于第三方的加密核，设计师没有任何办法优化 RTL 代码。除了等待 IP 供应商提供新版本 IP 核外，设计师唯一的选择就是在 FPGA 厂商提供的软件中使用高级的优化设置。设计师可尝试各种设置，直到该“动作”IP 核达到时序收敛的目标，然后锁定并保存该 IP 模块的布局布线结果。

若其他一些设计模块有变动，例如“顶层”模块的变动，它就不会引起“动作”模块和“控制”模块的时序收敛问题，因为它们已经被锁定了。

12.3.3 早期时序估计

正如第 8 章所提到的，除非设计已经进行了一定程度的布局，否则时序估计并不准确。因而在设计早期，设计师想通过完整的布局布线编译得到设计的性能估计是不可能的。如何在设计早期进行时序估计，FPGA 厂商已给出了解决方案。

大多数 FPGA 厂商提供的软件含有缩短编译时间的设置。通过限制布局尝试的次数能大大缩短编译时间，这通常以牺牲设计的性能为代价。使用快速编译选项获得的时序结果与使用完整编译时所获得的时序结果相比，通常误差范围在 10% 以内，但所用编译时间却不到一半。这的确是一个能大大缩短时序收敛周期的强大工具。

建议设计师在以下的场合中使用快速编译选项：

1. 在设计早期，当设计师确定要变动模块的性能时使用快速编译。所获得的时序结果可能在最终结果的 10% 误差以内，但迭代时间则会短得多。

2. 对容易满足时序要求的完整设计使用快速编译。如果设计相比目标 FPGA 技术而言性能并不高，这种编译方式会减少整个工程期内的设计迭代时间。

工程文档应该反映这个设计或某一设计模块所使用的布局布线选项。

如果设计的时序差 10% 以上才收敛，就应该返回去修改 RTL 代码，而不是继续进行完整的编译。

正如设计规划中所指出的，为了能尽早地发现设计集成和资源利用中的问题，设计师应该尽早地在顶层设计中编译主要的设计模块。为了做到这一点，设计师可以给未完成的模块创建虚拟模块。但这些空模块需要包含正确的接口。

12.3.4 CAD 工具设置

建议设计师尽量保持缺省的综合和布局布线设置。FPGA 厂商提供了几十个会影响时序结果的按钮和开关。设计时应该尽量避免乱动它们，只在对 RTL 编码已经无能为力时才使用它们。

这就是说，这些设置会非常有效并且能大幅度地改变编译结果。然而 FPGA 厂商所提供软件的各个版本给出的时序结果会截然不同。因此，这些设置会使设计在工具的各个版本间使用起来不方便，事实上它们会使所设计的 IP 不可重用。

如果设计师走投无路，而又不得不惜任何代价达到时序收敛，那么就应该充分使用这些选项。

除了优化设置外，FPGA 厂商提供的软件还能通过对逻辑进行版图规划来影响时序结果。设计师可以指定逻辑单元放置在不同的组、区域或者下至个别的布线轨迹。

再次建议设计师不要这样做，除非 FPGA 厂商提供的软件在布局上做得非常差。

人类架构专家几乎不可能用人工方法击败自动综合布局布线工具。然而在个别的孤立场合，当所有希望似乎都破灭时，人工架构方法也不失为设计师武库中的另一种武器。

12.3.4.1 理解布局布线工具

源自主流 FPGA 厂商的布局布线工具为了满足设计需求会调整其操作。换言之，基于相同的时序约束，设计师可能得到不同的结果。时序约束越严格需要的编译时间越长。

布局布线工具是由时序驱动的，因而它能理解复杂的时序约束。因此建议设计师们使用真实的时序约束。

布局布线工具会努力寻找一个能满足时序需求的布局布线方案。

FPGA 布局布线工具的特殊现象之一，是由其完成的具体布局和连接时，会根据“种子效应”产生不同的结果。

逻辑的初始布局是随机的，随设计的起始条件不同而有所变化，所以有可能产生满足设计目标的不同布局。布局布线的种子（也叫适配器种子）可改变布局算法的起点，这会对如何优化的过程产生影响。为了实现时序收敛的目标，布局布线算法会根据先前运行的结果，进行多种布局尝试。然而，初始布局的改变可能导致最终布局的不同，从而产生不同的时序。

"种子扫描"是用于时序收敛的一项常用技术。"种子扫描"通过运行多个不同的种子来确定哪一个种子能为设计带来最好的效果。过去，种子扫描会导致性能上的大幅变化。如今，种子扫描对最新 FPGA 技术性能带来的平均变化在 ±5% 的范围以内。注意对不同的 FPGA 厂商以及不同的器件系列，这种变化会有较大的差异。

对于打算重用的设计模块或者可能要求在今后进行更新的最终设计，建议设计师避免使用种子扫描。这是因为在 FPGA 厂商所提供软件的未来版本中，相同的种子会带来不同的效果。再者，若设计师对设计做任何改动，例如逻辑上的改动，配置上的改动或引脚上的改动，相同的种子也会带来不同的效果。

那么，什么时候使用种子呢?

1. 如果设计能满足时序，设计师还希望能将设计的时序裕量最大化。

2. 为了在实验室中对设计的功能进行检查，设计师必须马上得到设计样机。此时，不必使用特定种子或种子扫描进行布局，因为在功能检查基本合格后，设计者可随时使用种子扫描做布局优化，以提高时序裕量。

3. 使用种子是满足时序需求的唯一方法。设计不再推出新版本，这是设计的最终版本。例如 ASIC 设计的 FPGA 样机应使用种子扫描做布局优化，以提高设计的速度性能。

若实现时序收敛的目标必须依赖于某个特殊的种子，换言之，依赖于 FPGA 厂商所提供软件的某个特殊版本，则该 IP 或者设计模块就不可重用。

12.3.4.2　高级优化（对设计有更高要求时才选用）

正如在 CAD 工具设置部分中所提到的，FPGA 设计工具提供了几十个用于设计优化的选项。本节将介绍几个最具代表性、最有效的选项。

物理综合优化

大多数 FPGA 厂商提供的工具包含物理综合优化选项。物理综合紧密结合布局布线工具，会对时序有问题的逻辑再做一次综合。常用技术包括寄存器重新定时和寄存器复制。这两个技术也可通过修改 RTL 代码实现，但是需要重新编写大量的代码。物理综合还可以进行许多种别的优化，但是这两个优化是最常用的而且通常是最有效的。

在某些设计中，物理综合能把设计的时钟频率提高 20% 以上。对于那些精心编码合理使用寄存器的设计，其性能增益可能只能达到 1% ~2%。然而，这

种优化是要付出代价的。换言之，设计的编译时间会急剧增加，通常是2倍或者更多。它也会限制设计师使用形式验证工具，因为这些工具通常与寄存器重定时有冲突。

由于编译时间的影响，设计师应该考虑把物理综合的使用限制在问题模块的增量设计流程中。

使用物理综合是全自动的，即设计师只需要设置选项并编译。

设计空间的拓展

大多数FPGA厂商在它们的工具包中提供了一些实用程序。这些实用程序可以使用不同的设置和种子，自动运行多个编译，以期找到能为设计带来最好效果的设置。

由于种子对布局布线的影响，设计师只应在设计的最后阶段，当设计实际上已经完整并且设计师致力于时序收敛时，使用设计空间拓展。

这类实用程序通常能够执行十个或者更多的编译，因此将需要花费几天时间才能完成编译。

幸运的是主流FPGA厂商已将多处理技术应用于这些实用程序，因此多个编译能并行地而不是顺序地执行。这大大地缩短了编译时间。

使用设计空间拓展工具的缺点是，如果设计师修改了设计的RTL代码，由于种子的随机特性，设计师需要重新运行该实用程序。

设计空间拓展也能在设计中的单个模块上运行。这项技术很强大，可以缩短编译时间，并只对设计性能关键的区域进行最优化。

这项技术在增量编译设计流程中特别有效。在该流程中，设计空间拓展只对时序临界的设计模块进行优化。

如果设计师对一个设计模块或者完整的设计使用设计空间拓展，所使用的确切设置应该在设计中进行记录，以使其他设计师能重现结果。

12.3.4.3 编译报告和分析工具

审查综合和布局布线报告的警告有助于实现时序收敛的目标。这些警告常常提供了可用来帮助提高设计性能的信息。应该规定设计师在设计过程中必须无一例外地审查并消除来自工程的所有警告。这项工作是必须的，因为这些警告可能指出了设计中的问题，例如不恰当地使用了锁存器或者找不到时序约束等等。若警告来自所购买的IP，则设计师就不能通过修改RTL代码来消除该警告，这是警告审查面临的难题之一。若发生这种情况，则设计师应该与该IP供应商取得联系，若他们能证实该警告没有问题可以忽略，设计师就可以在工程中记录该信息，并且在以后的编译中忽略该警告。

报告文件本身有详细的关于FPGA器件资源使用情况的信息，从而可以用来确定器件中哪个模块使用的资源最多。

来自编译报告的信息有助于确定布局布线中的困难，例如消耗在布局布线上的时间。布线时间很长可能是由于布局不合理造成的。对某些节点做人工手动布局或在布局上多花些功夫，布线时间有可能缩短。

编译报告还提供已执行优化的详细资料，如已经从设计中移除的寄存器。这些信息能帮助设计师找到 RTL 代码中的问题，或解释为什么故障逻辑已经被移除了，使得设计师有目的地修改 RTL 代码。

编译报告还列出了被忽略配置的有关信息，设计者能从中分辨出哪些信息是创建配置时由打字错误而引起的，哪些信息是因为配置已过时而引起的，这些问题都应从工程中删除。

除了编译报告文件之外，FPGA 厂商还提供了用图形来描述设计的工具。

为了更好地理解 RTL 代码并查看综合和布局布线的结果，设计师应该用这些图形工具来仔细检查结果。

这些浏览工具提供了设计层次化的结构框图以及设计技术实现的视图。技术实现视图详细地描述了在综合或布局布线之后，设计是如何映射到目标技术上去的。

层次化的结构框图视图有助于理解设计的架构，因此也有助于理解如图 12-11 中所示的设计流。

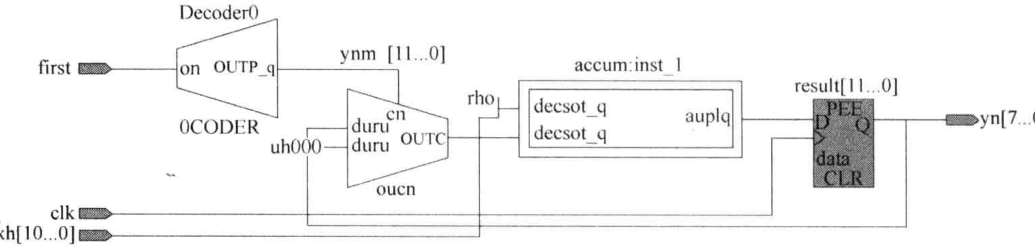
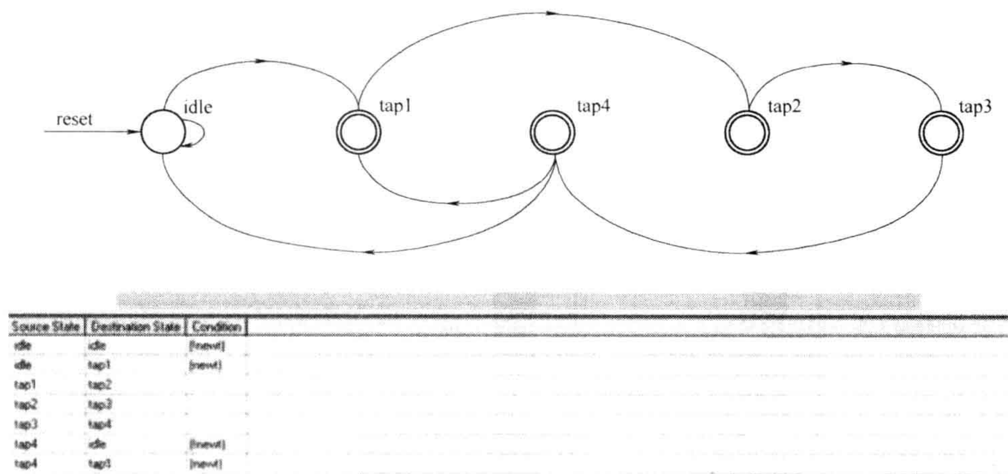


图 12-11　Quartus Ⅱ软件中的 RTL 浏览器示例

当从别处继承设计模块时，应该使用这些浏览工具对该设计有一个可视化的理解。由于浏览工具详细地描述了设计中的数据流和模块间的交互，有助于对器件进行版图规划。浏览工具还提供了可视的功能框图，如图 12-12 所示的有限状态机。

具体的技术视图有助于理解设计是如何在 FPGA 中实现的，并能用来确定可以被优化的区域。它能快速地给出关键路径上的逻辑层次数，并能链接到相关的 RTL 代码，有助于将 RTL 源代码与具体电路的实现关联起来。

技术映射视图和时序分析工具一起使用时，有助于为设计创建合法的复杂时序约束。时序分析报告中的每一条路径可以在技术映射视图中定位。在技术映射视图中，设计师可以仔细检查这条路径的实现，确定该路径是否属于时序异常，



Source State	Destination State	Condition
idle	idle	(!newt)
idle	tap1	(newt)
tap1	tap2	
tap2	tap3	
tap3	tap4	
tap4	idle	(!newt)
tap4	tap3	(newt)

图 12-12 在 Quartus Ⅱ中来自 RTL 浏览器的有限状态机示例

例如多周期路或伪路径，然后在时序约束文件中做出合理的分配。

12.3.4.4 版图规划工具

所有 FPGA 厂商提供的设计工具都包含一个版图规划工具，在某些情况下还会有多个版图规划工具。

在 FPGA 时代的早期，这些工具曾在理解 FPGA 设计的架构和优化设计性能方面起过决定性的作用。

如今，前一种说法即有助于理解 FPGA 设计架构的这种说法依然正确。版图规划工具有助于解释 FPGA 器件中什么资源可用，并可用于分析设计的布局布线结果。但是，后一种有关优化设计性能的说法就不那么正确了。在大多数情况下，没必要通过设计的版图规划来达到性能要求。要说版图规划能为性能带来好处，可能也就是针对小部分设计的版图规划，而不是针对整个设计。

如今版图规划在自底向上的设计流程中也有帮助。在这种场合中，设计师只要将设计模块分配到器件的各个区域即可，不必操心 FPGA 的单元级的设计。只要把每个主要的设计模块分配到器件的某个区域即可。

总而言之，FPGA 厂商的版图规划工具有四个主要用途：架构探索、布局布线分析、创建布局分配和工程设计更改。以下详述这四种用途。

架构探索

版图规划提供了芯片资源的可视化显示。它类似于在设计师的桌面有一个详细的已用资源和未用资源的数据一览表。版图规划可用来查看 FPGA 器件架构的细节，如在一个逻辑阵列块（LAB）中寄存器的数量，在一行中 LAB 的数量，存储器的布局以及布线信息。它还允许设计师查看专用模块的内部逻辑，如 LUT

的配置和寄存器。

图 12-13　Quartus Ⅱ技术映射浏览器中的关键路径视图

图 12-14　Quartus Ⅱ芯片规划工具中 Stratix Ⅳ自适应逻辑模块架构的细节

版图规划可提供用图形表示的 I/O 单元配置（例如延迟链、I/O 标准、信号方向）和 I/O 单元内部寄存器使用等情况的详细资料。

查看多个设计模块的连接是版图规划在基于团队的设计中的实际用途。

版图规划对时钟网络的规划也特别有用。版图规划在详细描述锁相环（PLL）配置的同时，也详细描述了芯片中哪些区域能由 PLL 的输出驱动，哪些区域能由器件的全局时钟信号驱动。这种功能在基于团队的设计环境中非常好用。在这种设计环境中，设计师需要给不同的工程师和功能模块分配资源，以防

止资源冲突并使设计师能针对资源的共享和整合进行规划，例如PLL。

布局布线分析

版图规划工具为检查设计的实现提供了一个很好的解决方法。它显示了逻辑的布局信息、详细的布线信息、扇入和扇出连接，并且还可以查看关键路径信息。

只有在设计存在问题时才需要进行布局布线分析。在时序出错时，布局布线分析可以与时序分析器配合使用，先将时序报告中的失效路径在版图规划视图中进行定位。然后才可能去分析设计的布局布线，以确定能否通过位置约束来修正时序，解决时序问题，或用图示表明芯片中某个区域已十分拥塞。

用版图规划工具可以清晰地观察到寄存器间的逻辑层次数量和I/O单元中是否使用了寄存器。该信息也能用其他工具查看，如编译报告和技术映射视图。

版图分配

版图分配工具可通过布局分配优化设计的性能。在大多数情况下，人工布局不如自动布局布线软件。然而，某些情况下，人工布局也是有用的。例如在访问专用硬件模块和/或引脚时，由于资源受限，节点间流水线寄存器的放置相距甚远。在这种场合下，布局布线软件不会总在源节点和目的节点之间优化寄存器的布局。为了优化布局和性能，设计师可以在版图规划中移动寄存器。

在版图规划中区域分配主要被用于创建区域约束，在增量设计或基于团队设计的环境中常用区域分配划分区域。这种场合中，首先在版图规划中创建区域，然后将设计模块分配到该区域中。换言之，使用区域分配能防止某个区域内的资源被别人占用，为尚未完成的设计模块保留可用资源。

在创建区域分配时遇到的难题之一，是如何处理内部存储器模块和DSP模块。根据模块的资源要求，为使设计包含足够的存储模块或DSP模块，设计师可能需要创建一个非矩形区域。

设计师还必须考虑所设计的模块如何与设计的其余部分接口，以避免无意中破坏了时序的收敛。

工程更改指令

版图规划工具提供了设计快速小修改的手段，因而能缩短系统在线调试所需时间。

它允许在设计中编辑、创建和删除逻辑及连线。建议设计师只做简单的更改，例如改变时钟的极性、时钟使能、或者插入简单的测试逻辑等。

这种方法对改变I/O单元的属性特别有效，例如改变延迟链的值、上拉电阻的使用、转换速度、I/O标准和电流强度等。

它还可应用于修改PLL设置或者将一个信号连到引脚进行分析。

不推荐设计师在投入生产时使用这种方法来改变逻辑，因为这样做RTL代

码不再和实现的功能相匹配。该方法只能用做简单的更改，还要能证明更改后能在系统中能正常工作。进行工程更改后需要做的工作有：对 RTL 代码进行修改以匹配功能、对设计进行仿真、重新编译并产生新的映像文件用于系统在线测试。除此之外，还应当对新版本设计进行完整的验证。

12.4　常见的时序收敛问题

本节列出了设计师可能会遇到的一些常见的时序收敛问题，以及如何解决这些问题而应当采取的措施。

12.4.1　缺失时序约束

FPGA 厂商提供的布局布线软件基于所给出的时序约束优化设计。若设计师没有对某关键路径进行约束，则该路径就不能被 FPGA 软件优化，因此该路径就可能不满足时序。对于更复杂的问题，设计师可能还不知道该设计存在时序问题。时序分析只会报告不满足时序约束的时序，因此若某路径没有被约束，则该路径就不会被分析。

大多数时序分析工具有一条命令来报告没有被时序约束的路径。建议设计师执行该命令来查看设计中是否有未被约束的路径，然后对这些路径设置恰当的时序约束。

设计师使用正确的时序约束十分重要。要分析时序报告并确认所有的多周期路径或伪路径的确是时序异常。使用通配符作为时序异常约束配置的一部分是很容易的，但这样做会在不经意间将该约束应用在不是时序异常的寄存器上，从而会导致在线测试过程时发现时序出错，而该错误在时序分析文件中却无报告。

12.4.2　时序约束发生冲突

设计师使用通配符设置时序约束有可能引起路径的时序约束冲突。虽然鼓励使用通配符，但设计师必须确定所使用的通配符是正确的。若某路径上存在多个互相冲突的约束，则布局布线引擎只能按照其中一个约束条件进行优化。该约束条件通常是最后输入的约束。这将导致其他约束不起作用，从而造成时序错误。

时序冲突通常发生在有多时钟域路径的设计中。

12.4.3　高扇出寄存器

高扇出寄存器的目的寄存器位置会导致源寄存器与目的寄存器之间的长线延迟。布局布线软件通常会进行布局优化，因而这也不是个问题。然而当位置约束限制了布局的选择时，这就是个问题了。例如一个高扇出寄存器是很多寄存器的

输入，这些寄存器又连接到 FPGA 器件不同面的引脚上，由于这些寄存器到引脚的 t_{co}要求非常临界。为了满足 t_{co}时序，不得不把目的寄存器放置在 I/O 单元内或者紧邻 I/O 单元。而将源寄存器放置在所有目的寄存器附近是不可能的。

以下是解决这个问题的最好方法，采用其中一个即可：

1. 创建更好的引脚分配，或

2. 复制源寄存器，使它们能靠近每组引脚放置。最好在 RTL 级进行这个操作。

12.4.4 只差一点就能满足时序

若设计已基本完成，但还差一点才能满足理想的时序要求，并且设计进度已不允许设计师再返回到 RTL 代码进行修改，则设计师应该尝试 FPGA 设计工具中每一个可用的选项，努力达到时序收敛的最高目标。大多数厂商提供的软件有设计空间拓展的功能。这种功能除了使用种子扫描布线外，会循环变化各种优化设置来尽量找到满足时序要求的最优设置。该方法特别耗时，因为设计师可能要运行 10 多次编译。然而，它可能带来超过 20% 的性能改善。为了缩短多次编译所需要的时间，设计师应该使用设计空间拓展工具中的高级功能，在多台机器上并行执行多设置编译。

12.4.5 不宜过早设置位置约束

若在设计过程的早期就设置位置约束，则在整个设计演变过程中通常会保留这个约束的设置。然而过早将位置约束值添加到设计中，有可能阻碍后期设计版本的性能提升。

还有一种诱惑就是过度地约束设计。虽然位置约束对单个模块的效果不错，但在设计整合后就会限制布局布线工具能进行的优化，从而导致性能变差。

在以上这两种情况中，建议删除逻辑位置约束，创建原设计的新版本。若设计不能满足时序要求，则检查哪个模块有问题，对出现时序问题的模块，恢复其原来的位置约束。看看时序是否因删除该约束而受到影响。若对时序没有影响则删除该约束。若对时序有影响，则保持该约束，然后针对其他约束进行同样的工作。

理想情况下，设计师希望不使用逻辑位置约束就能实现时序收敛目标。

12.4.6 冗长的编译时间

避免冗长编译时间的第一种技术是使用增量编译设计流程。若设计师使用增量编译方法，则不会遭遇冗长的编译时间。

避免冗长编译时间的第二种技术其实是对第一种技术的补充。就是使用带有

多处理器或多核处理器的工作站。FPGA 厂商所提供设计软件的算法是多线程的并且能充分利用多核或处理器来缩短编译时间。为了充分利用多处理器，设计师应该确保该工作站有大量的高速 RAM。面向最新 FPGA 器件设计的编译，可能要使用多达 16G 的 RAM。这些算法不断地访问 RAM，因而高速 RAM 有助于缩短编译时间。

如果设计很容易满足性能要求，设计师可以考虑使用 FPGA 厂商提供这两个选项之一来快速地对设计进行布局布线。这样可以减少一半的编译时间，但可能导致设计性能的降低。

12.5 设计规划、实现、优化和时序收敛清单

1. 遵循同步设计原则；
2. 遵循推荐的编码规范；
3. 为增量设计进行设计划分；
4. 保证 RTL 代码可充分利用器件中的专用硬件资源。

在 RTL 代码中实例引用厂商提供的原语模块，就可以利用（不能经由 RTL 代码综合得到的）由该模块所描述硬件的特殊功能。

5. 为设计创建完整的时序分配；
6. 确保所有的多处理器功能已经开启，以缩短编译时间；
7. 对设计中时序临界的区域进行版图规划；
8. 对工艺的所有边角条件进行时序分析；
9. 分析所有错误和警告，做出必要的改动以消除警告并记录所有的异常；
10. 记录达到时序收敛目标时的所有设置。

第13章　系统在线调试

13.1　系统在线调试的难点

对在系统中运行着的任何芯片进行调试都十分困难，调试者经常会被搞得精疲力尽。你设计的电路板也许能正常工作，也许突然纹丝不动。这时设计师的脑海里就会出现这样的想法："我的设计能正常工作吗?"。于是工程师们开始聚在一起讨论：问题究竟出自系统的软件还是系统的硬件？由于系统软件的开发费用十分昂贵，所以一开始几乎总是怀疑问题来自硬件，除非能证明硬件确实没有问题。本章中，我们将关注能用于快速故障定位的技术。

就系统在线调试而言，FPGA 与 ASIC 相比具有明显的优势。这个优势就是 FPGA 的可编程性。对 ASIC 设计而言，为了验证设计能在电路板上的正确运行，设计者不得不先设计调试逻辑；为了避免再次投片的昂贵费用，ASIC 设计者还需要尽可能保证设计功能 100% 地正确。在做 FPGA 设计时，预先设计的片内调试逻辑是设计师必然用到的关键功能；然而 FPGA 固有的可编程性，使得调试逻辑可以由主机控制，或随着系统在线调试的进展，被添加到设计中。

仿真的目的是在设计载入芯片之前，能够发现设计或模块集成中的一些错误。然而，对 FPGA 设计进行穷举性的仿真非常耗时而且计算量十分巨大。若能用真实的条件对设计进行测试，则可以发现仿真难以发现的问题。这样的例子有异步时序问题、信号完整性的特殊问题和软/硬件集成问题。

本章中，我们将推荐一套调试方法，使设计者能按自己的意愿观察系统的在线运行情况，并帮助设计者发现系统在线运行时存在的问题，以验证设计是否正确。本章讨论的技术源于目前常用的工具和技术。

13.2　规划

在创建设计时，大多数工程师往往没有考虑到设计或实现中会有错误。缺乏经验的工程师只有在电路板测试出现问题时，才开始考虑系统的在线调试。而经历丰富的工程师由于已多次承受了设计调试的压力，从而希望减少在高度紧张环境下调试的时间。她/他不想耗费无数个夜晚或周末在实验室里为出现的问题找原因。因此，这些工程师会事先编写调试计划。这也是每个设计者需要做的！

系统在线调试计划应该作为设计说明书的一部分认真地编写。设计中的每个主要模块都应该有一个在线调试计划，即如何验证模块的在线运行情况，以及针对该模块应使用哪种调试策略。设计说明书还应包含关于判断该模块是否已按预期运行所依据的信息类别。换言之，判断模块功能是否正确还应依据以下三种信息：

1）系统级别的统计信息，例如存储器的接口效率；

2）总线上的性能瓶颈分析；

3）高速收发接口的比特错误率信息。（译者注：至少根据这三种信息才能判断模块在线运行是否已经达标。）

除调试模块之外，在所有的设计模块都实现之后，对顶层设计也应该有一个调试计划。这些信息源自第 4 章，其主要内容介绍设计的密度和引脚。

调试计划应该指明为系统在线调试预留了多少只引脚、多少逻辑和存储器，还应详细列出系统在线调试过程中的使用方法和工具。

为调试设计预留 15% 的器件引脚是一个好方针。这不包括用于加载 FPGA 器件所使用的 JTAG 引脚和部分调试过程所使用的引脚。推荐的调试资源需求将在第 13. 3 节（关于调试方法这一节）中将进一步讨论。

13. 3　调试方法

为方便系统设计的在线调试，FPGA 厂商和 EDA 公司提供了多种可用工具。在本节中，我们将介绍最常用的工具和方法，以及在什么时候使用它们。

13. 3. 1　利用引脚调试

这是针对 FPGA 设计最常用的调试方法。它之所以很受工程师欢迎是因为可以通过编程将不同的信号很方便地布线至 FPGA 器件的各个引脚，而且软件处理引脚布线的编译时间很短。这样，在实验室中进行调试时，在几十分钟之内就能根据新编写的文件将不同的信号连接到调试引脚。这种方法，除了会增加所检测信号的扇出之外，在大多数情况下不会影响原设计的实现。

若你的设计已几乎把 FPGA 的资源耗尽，而此刻还必须对布局布线做改变，才能够把某几个信号连接到引脚。在这种情况下再进行布局布线，很可能产生异步时序问题，必须避免这种情况的发生。由于此刻资源缺乏，布局布线工具无法将外部输入的异步信号同步化。[译者注：此时综合器将不对异步信号做同步化处理。]

正因为如此，想通过引脚进行调试，设计者必须预先为调试留好选定的引脚或引脚区域。

在 FPGA 设计软件中有几种方法可将内部信号布线连接至引脚。最常用的方法就是通过 Floorplan 工具，在里面选择所需信号作为源，选择相应的引脚作为目的地，布局布线软件会使用增量布线将信号连接至该引脚。这种方法用于一两个信号的连接时确实很简单。然而，对比较大的信号组采用该方法就会变得很费劲。举一个常见的例子：在 32 个引脚上调试 32 位总线。当然，有些工具有高级功能，它允许设计者通过信号的查找功能或者脚本化的接口来选择信号源和目的引脚，然后自动地实现信号和引脚的连接。

对引脚信号的布线时序要求非常重要，特别是将总线信号布线连出至相应引脚时。为了使总线和时钟同步，建议将信号在引脚的寄存器上锁存后再输出。你不希望这些信号成为设计中的关键路径，因此必须对这些路径添加时序约束。对于高性能的设计，设计者可能需要在信号和引脚之间插入几级流水寄存器。一些 FPGA 设计软件已经提供了这个自动化选项。

使用引脚调试信号的步骤如下：

1. 为调试预留一些引脚；
2. 在这些引脚上设置恰当的 I/O 标准；
3. 确定设计者希望连接到某引脚的信号；
4. 确定某信号是否需要插入流水寄存器；
5. 进行恰当的时序分配；
6. 将某个信号布线连接至某个引脚；
7. 分析该信号的时序；
8. 对 FPGA 器件进行编程；
9. 用外部逻辑分析器或示波器分析这些引脚上的数据。

如果设计者想要在某个引脚上查看不同的信号，可以删除与该引脚连接的不再需要查看的信号，然后从步骤 3 开始重复。

13.3.2　片内逻辑分析仪

片内逻辑分析仪（ILA）这个调试工具已帮助许多设计者发现了许多故障，为他们排忧解难。起初很多设计者认为 ILA 只是设计流程中的一个可选项而已，直到有一天，遇到一个在仿真中发现不了的设计漏洞，此时他才想起使用 ILA。他用 ILA 隔离故障，找出问题，修复系统并加以验证。在见识了 ILA 的强大功能后，ILA 就成为工程师在 FPGA 设计流程中的得力助手。

主要的 FPGA 厂商和一些 EDA 工具商都提供 ILA，ILA 是利用 FPGA 器件内部空余的逻辑和存储器资源实现的。

究竟什么是 ILA 呢？

其实 ILA 是一种用 FPGA 内部逻辑单元实现的片内调试工具，它可提供与普

通逻辑分析仪类似的触发能力，并观察设计内部多路数字信号。ILA 的优势在于不必为调试多保留若干个引脚，只要通过 FPGA 上专为 JTAG 接口保留的引脚就可以输出需要观察的多路信号。采用最新技术制造的 FPGA 器件，其运行时钟频率可高达 250MHz，即使在这样高的时钟频率下，ILA 也能捕获到内部信号的数据。然而，ILA 的性能会随所选用的触发条件的复杂度而有所改变。使用 ILA 时不必改写设计文件，这是 ILA 的另一个优点，这是因为 FPGA 厂商提供的软件能自动地将 ILA 插入到已实现的 FPGA 设计中。对设计的实现无任何影响。

ILA 所捕获的信号数据被保存在 FPGA 片内存储块中，设计者可以随时读出并分析这些数据。另外，在一个 FPGA 芯片中还可以实现多个逻辑分析仪，这样做的好处是能同时从设计中的不同时钟域捕获数据。

既然 ILA 如此之好，那么请问，为什么并非所有的设计师都使用 ILA 呢?

答案十分简单：设计计划做得不够好。在很多设计中没有为使用 ILA 保留足够多的 FPGA 资源。最常见的失误是没有保留足够多的存储器资源以保存需要分析的数据。

本书曾多次提到，设计者必须提前规划调试所需资源。

为了使用 ILA，设计者必须确保以下 3 个条件：

1. 有可用的 JTAG 连接；
2. 保存分析数据的存储模块足够大；
3. 有产生触发条件的逻辑。

大多数 ILA 均有如下标准功能：

1. 样本深度和用于保存采样数据的 RAM 类型均可由调试者设置；
2. 高级触发条件，例如基于状态的触发。高级触发条件精确地定义了在什么情况下，（ILA）将开始采集数据；
3. 连续数据的存储。当触发条件发生时，采集到的数据被连续地写入存储器。为了防止已保存的数据被覆盖，这种操作模式需要大量的内部存储器；
4. 变化数据的存储。在数据采集的过程中，当采样值与上一个采样周期的采样值有所不同时，才将新采样值写入采集缓冲区。若采样值没有发生任何改变，则下一个采样周期不保存该采样值。
5. 条件性存储。只有当把数据写入存储器的条件成立时，才存储数据。

实现 ILA 所必须的逻辑元件和存储器的数量取决于触发条件的复杂度和需要保存的数据量。

减少所需逻辑元件数量的有效方法是将采集缓冲区的段数减少到只要能满足调试需求即可。

另一种技术是使用缓冲采集控制来精确地控制写入采集缓冲区的数据。若采用这种技术可以丢弃与该设计调试无关的采样数据。变值存储和条件性存储可用

于减少必须的内部存储器的数量。

13.3.2.1　使用ILA的设计流程

1. 在设计中添加ILA。该功能由FPGA厂商提供的软件自动插入，不用修改设计代码，也不用修改FPGA器件中已实现的设计；
2. 配置逻辑分析仪。定义希望观察到的信号，设置记录信号值的条件；
3. 定义触发条件；
4. 编译设计；
5. 对器件进行编程；
6. 在主机上运行ILA应用程序；
7. 查看并分析所捕获的数据。

13.3.2.2　ILA的局限

由于FPGA结构的限制，并非设计中所有的信号都能被ILA观察到或者引出，例如进位链中的部分信号，就无法用ILA观察；JTAG信号也无法观察。

除非设计者愿意重新编译整个设计，否则设计者只能查看到布局布线之后的可见信号。因为RTL级综合过程中进行的优化往往会改变信号的名称，这使得设计中的组合信号难以识别。要使这些信号可见，设计者可以在RTL代码中使用综合属性来保留这些信号。但这样的设置会改变设计的实现。因此，建议设计者集中精力对寄存器进行系统在线调试，因为大部分寄存器在布局布线后是可见的，并且不需要重新编译整个设计。

13.3.2.3　小建议

远程调试

如果在最终设计中保留ILA，并且有JTAG连接到FPGA，我们就能对设计进行远程调试。对异地的设计进行调试，远程调试被证明是非常有价值的，甚至设计者在办公室或家中也能调试实验室中设计；但前提条件是设计者和电路板相连的工作站之间能通过网络通信。

与MATLAB的接口

一些更先进的ILA提供了与Mathworks MATLAB软件的接口。这个选项对数据进行数字信号处理（DSP）分析很有用。一旦数据被导入MATLAB环境中，数据就能以适合用户应用测试的格式展现出来。

器件资源不足

如果设计中未留有可构造ILA的足够资源，那么在线调试就有困难，所以必须从最终的设计中删除一部分功能再继续调试，这也是调试工作的一部分。这样做可以使设计者在线调试独立的模块，并验证这几个关键模块的功能是否正确。虽然这样做不能解决整个系统在集成中出现的问题，但使设计者能检验这几个关键模块在集成中出现的问题。

13.3.3 调试逻辑的使用

在设计中插入调试逻辑是实际工作中常用且推荐的设计方法。第 13.4.3 节将讨论这个问题，并给出系统性能报告。

正如以前所提到的，设计者应该针对主要设计模块的接口构建测试逻辑、监测器和验证器。在证实设计的功能完善之后，可以删除调试逻辑；也可以在设计中保留调试逻辑，以便在现场出现故障时，可提供远程调试能力。若调试逻辑被保留在所设计的产品中，则建议调试逻辑可以由引脚、JTAG 或软处理器关闭或控制，这将减少最终设计的功耗。

调试逻辑也可以和本章描述的其他调试方法一起使用。添加一个简单的与调试引脚接口的多路器，便可以使用户很容易地将他们希望查看的信号连接到引脚上，用户或软处理器可以自由地切换与调试引脚连接的信号，观察其波形。当设计者需要查看不同信号的时，就能快速地将信号切换到调试引脚，而不必每次都产生新的 FPGA 编程文件。这种方法可以节约好多调试时间。

调试逻辑也可以用于强制 FPGA 进入某种特定的状态，以再现故障条件或者测试在这些隔离边角条件下的运行情况。

主流 FPGA 厂商提供了实用的调试工具，利用这些工具可以强制逻辑进入某个特定的状态。使用这种实用调试工具能显著减少开发工作量。

再次强调一下，这些工具可以与其他调试方法相互组合，形成高级调试方案。比如，与 JTAG 相结合时，能使设计者动态地控制实时运行的控制信号。同样，它也能与 ILA 相结合，强制产生触发条件。通过这些方法，就可以使用简单的测试向量来运行设计，并显示其内部信号的波形，而不需要使用外部测试设备。

13.3.4 外部逻辑分析仪

主流 FPGA 厂商提供与安捷伦（Agilent）和泰克（Tektronix）公司的逻辑分析仪的接口。为了在逻辑分析仪中使用这些可选的接口，用户需要有一个 JTAG 连接器和供逻辑分析仪使用的测试插口。

在 FPGA 设计全速运行的同时，该接口能够使用最少量的 FPGA 输入/输出（I/O）引脚，利用外部逻辑分析仪查看芯片内部信号。

与 13.3.3 节描述的调试逻辑方法类似，这种方法使用多路器将大量的 FPGA 器件内部信号连接到少量的输出引脚上。

多路器由 JTAG 通过逻辑分析仪的用户接口进行控制。并且为了简化调试，逻辑分析仪还能在屏幕上显示信号的名称。

与 ILA 相比，该调试方法有如下两个关键优势：

1. 更广泛的样本深度；

2. 更强的数据处理能力。外部逻辑分析仪拥有比 FPGA 器件内部数量多得多的存储器。

所以，当设计者需要存储和分析大量调试数据，并且已在电路板上为测试插口预留了空间时，推荐使用这种调试方法。

13.3.5　编辑存储器内容

在设计中，我们可以用内部存储模块的内容来强制系统进入测试和调试状态。该方法在测试数字信号处理应用时特别有效，例如滤波器中存储模块里面存储的滤波器系数。进行这个操作有如下三个主要方法：

1. 第一种方法：加载一个新的编程映像，更新存储器的初始化文件。设计者不需要重新编译设计，只要改变存储器的初始化文件，然后运行汇编生成新的编程映像即可。该方法虽然可行，但是改变存储器的内容需要给 FPGA 系统重新上电。

2. 第二种方法：通过生成逻辑使设计者能对内部存储器进行写操作，以来改变它的内容。这种方法我们在 13.3.3 节使用逻辑设计进行调试部分曾经描述过。由于能在设计运行的同时，控制存储器模块的写操作，这种方法比之前的方法要更灵活一些。可能控制逻辑的设计会十分复杂，但是其回报却是无法估计的。

3. 第三种方法：使用 FPGA 厂商提供的解决方案之一，就是利用 JTAG 接口来控制内部存储器模块的读写操作。这个方法需要遵循一定的设计原则：被修改内容的存储器，必须使用 FPGA 厂商提供的存储器原语来进行设计。这种方法能够最简单、最灵活地在系统中更新存储模块内容，但它仍有些局限性。其中最大的局限就是不能在双端口 RAM 上使用。

这些调试方法不仅对 DSP 应用，对于其他应用效果也非常好。

它们不仅可以用来测试和纠正存储器的奇偶校验位，还可以人为制造奇偶校验错误值，以检查设计对错误的处理能力。除此之外，如果在实验室中发现由于奇偶校验位错误导致系统故障，设计者能使用这种方法来纠正错误并且继续核查。

该方法可以与本章描述的其他调试方法相结合，为自己装备一个非常强大的调试工具武器库。

13.3.6　利用软核处理器进行调试

许多设计师忽略了这样一个现实的方法，就是可以在设计中添加软核处理器来进行设计调试。添加一个软处理器只要用 1000-2000 个逻辑元件，加上部分内

部存储资源。

当软核处理器与传统的调试逻辑相结合时，将会变成强有力的调试武器。处理器能控制调试逻辑的操作，其本身也可用作调试逻辑。描述复杂的调试触发条件将变得比较容易，例如用 C 语言来描述状态机的触发条件就比 HDL 容易得多。

软核处理器也可用于控制存储器的读写操作，它不但拥有片内逻辑分析仪（ILA）解决方案的所有优势，它还能将数据存储在例如 DDR Ⅲ这样的外部存储器中，这样就能存储大量的数据以供分析。

如果设计者喜欢用 C 语言来编程，那么就应该尝试一下使用软核处理器来调试设计。

13.4 使用案例

13.4.1 上电调试

当电路板第一次上电工作时，设计者首先要确认一下设计是否按照正确的顺序执行，以确信该 FPGA 设计能与系统其余部分正常通信。若上电后系统出现不能运行的情况，则立刻使用 ILA 捕捉 FPGA 在上电或复位后的 FPGA 器件初始化过程中的若干个触发事件。ILA 能在 FPGA 器件编程后即刻捕获数据。一些 FPGA 厂商已经提供了具有上电调试功能的 ILA 解决方案。

13.4.2 收发接口调试

在电路板上电之后，设计者总希望确定 FPGA 器件上的收发器是否能正常工作，例如是否能发送/接收来自系统的数据。

在实际应用中，收发器使用的设置与实际的电路板不完全匹配的情况比较常见。在这种情况下，如果收发器能重新动态配置，就可以在 FPGA 器件运行时对它的设置重新编程，那么调试就相当容易。幸运的是，主流 FPGA 厂商提供了这个解决方案，让用户能够完全访问收发器中的设置，并报告数据的误码率。

若在收发器接口中已构建了设计调试模块，设计者就可用现成的设计来测试收发器的误码率，也可以直接把厂商提供的某个调试设计下载到 FPGA 中来测试收发器的误码率。后一种方法最为常用。

这些调试设计由数据模式发生器、校验模块和收发器三部分组成，其中收发器可以动态重配以改变 PMA 的配置。在发射端，可以针对不同通道介质设置预加重值，它影响输出端差分电压（Differential Voltage，VOD）和接收端的眼图；在接收端能改变均衡和直流增益的设置。

通过循环设置、生成和校验数据，能在每个设置上得到误码率测试结果。这

样做的目的主要有两个：

1. 分析收发器的信号质量；

2. 调整收发器设置，以匹配承载的电路板并抑制收发器接口与电路板之间可能存在的信号完整性问题；

一旦找到了最优的设置，就可以将其应用于实际的收发器设计中。

13.4.3　系统性能报告

一般会收集设计中系统级别的统计信息来确定设计是否达到期望的系统性能。这些信息包括系统的吞吐量和带宽使用情况。通过这些信息，就可以找到设计中的瓶颈，并改善设计来满足对吞吐量和带宽的要求。使用监测器来完成收集和分析的工作。

在早期测试或者分离边角情况下，设计者可能需要去产生数据流来运行不同的传输情况。通常系统软件会负责这个工作，然而在电路板调试的初期，软件还存在问题或者它根本还没有准备好，硬件工程师就需要有一种手段来产生测试设计模块的数据流。

对于使用特定协议的应用，设计者可能希望检查并报告与协议冲突的地方，这需要测量并分析传输情况和信号。

这类数据的捕获、激励和报告，最好通过在设计中构建验证 IP 来解决，例如，能将处理器子系统模块挂起的监测器和接口 IP 上的协议检查器。

如前所述，通过计划系统在线验证，设计者在一拿到硬件时就能让设计正确地运行起来。如果像第九章推荐的那样，在设计模块中使用标准接口，就能快速构建一个验证 IP 库，该库还可以在以后的设计中重用，可以方便地插入所设计的系统中。设计者可使用系统集成工具，比如 Altera 的 SOPC，将验证模块拖曳到系统中，将设计的工作量最小化，而且对系统性能的影响也很小。同时将验证 IP 保留在最终设计中，对现场出现故障的任何系统的维修都十分有用。设计者使用的验证 IP 也可以和 FPGA 器件上进行 JTAG 控制的基础设施一起使用，这样设计者就能通过 JTAG 接口访问/控制数据。

13.4.4　软核处理器调试

软核处理器设计的调试，同时涉及硬件和应用软件的调试，使得调试过程十分复杂，这就要求工程师要熟知相关学科的知识。硬件调试可以通过使用本章先前介绍过的方法进行，之后还需要和运行在软核处理器上的代码一起执行。部分调试工作可使用强制硬件进入已知状态的方法，有效地模拟软件的运行。

软件调试很大程度上依赖于正在使用的软件工具链。建议设计者阅读软核处理器的文献，以熟悉可用的调试功能。

本章的其余部分，将关注在大多数软件调试工具链中存在的标准功能集，以及它们是如何用来进行设计的实时分析的。

13.4.4.1 软件分析

大多数处理器的工具链提供软件分析器。软件分析器可提供有关不同功能在应用中运行时间长短的报告。软件分析器能确定所设计的代码中可能会导致设计性能问题的非优化区域。设计者应该通过分析软件来决定需要优化的软件代码或可以通过硬件加速的代码。

13.4.4.2 观察点

在代码中插入观察点能捕获一个全局变量的所有变化。这种方法可以发现已被破坏的（“C”代码中的）全局变量，因此十分有用。

13.4.4.3 堆栈溢出

这种方法适用于当有实时操作系统在处理器上运行的情况。在这种场景下，每个正在运行的任务有它自己的堆栈。这增加了堆栈溢出状态发生的可能性。在基于 FPGA 的嵌入式系统中这类问题更加常见，因为它非常可能对堆栈可用的存储器数量进行限制。大多数处理器的集成开发环境（IDE）含有一个可选设置项，用于启动实时的堆栈检查。

13.4.4.4 断点

一些软核处理器的工具链提供了一个调试选项，对放置在只读存储器中的代码设置硬件断点，如闪存。这需要修改代码的编译设置，虽然会导致可优化的代码减少，但代码调试起来却容易得多。

13.4.4.5 单步执行代码

将软件编译器的优化级别设置为无，虽然软件代码会运行得慢一些，但由于源代码和执行代码完全匹配，调试工作就会容易得多。这种方法和软件断点配合时效果很好：执行代码会一直运行到断点处，然后就停止在那儿。这种方式可以支持单步执行代码，检查设计中的变量值，验证执行代码功能的准确性。

13.4.5 器件的编程问题

第三方公司和 FPGA 厂商提供了大量的 JTAG 调试工具，帮助设计者通过 JTAG 调试 FPGA 器件的编程问题。最常见的难题是企图调试来自不同厂商的 FPGA 器件的 JTAG 链上出现的问题。

来自 FPGA 厂商的调试工具可以专门测试 JTAG 链的信号完整性以及检测 JTAG 链的间歇性故障。这些工具检查 FPGA 器件连接是否正确，并能运行 JTAG 的调试命令。

这些工具在检测以下故障时非常有效：

1. 开路；

2. 与 Vcc 短路；

3. 与 GND 短路。

建议设计者在接收到电路板后，立刻使用 JTAG 链上的 JTAG 调试工具对电路板进行上述 3 项检查。

13.5　系统在线调试核对清单

1. 调试规划

（a）为调试预留引脚；

（b）为使用片内逻辑分析仪（ILA）预留逻辑资源和存储器资源；

（c）确保所使用的 JTAG 能够访问到 FPGA 器件；

（d）在电路板上放置一个测试插口，用来与逻辑分析仪或者示波器连接；

（e）向设计中添加调试逻辑，或者使用 FPGA 厂商提供的调试工具将数据存入存储器，或者将数据经多路器连接到引脚；

（f）考虑在设计中添加软核处理器用于调试；

2. 进行调试

（a）使用增量编译锁定原有设计的实现；

（b）将实时运行的数据信号或少量控制信号通过增量布线连接至引脚，方便在逻辑分析仪或示波器上进行分析；

（c）为了捕获基于事件触发的数据，可以在设计中添加一个片内逻辑分析仪（ILA）。如有可能的话，使用布局布线后的信号名，以避免对该整个设计进行重新编译。

3. 若在 JTAG 链中有多个 FPGA 器件，则选择需要调试的器件。

4. 一旦找到了错误，马上修改 RTL 代码，并且通过仿真验证修改是否正确。

第 14 章　设计的签收

14.1　设计签收过程

设计流程中需要安排一个步骤来决定设计投片的时间节点。在设计已通过完整的硬件测试，并且所有的设计和测试过程都已达标之后，再做这个决定。

应该召集项目的所有股东召开一个管理会议，来决定是否投片。会议上将审核设计的质量资料，由此决定设计是否应投入生产。

设计中所有已发现的错误应该得到纠正，但对产品发布而言非关键性的错误也可以被接纳。被接纳的错误应该用文档加以说明，以便到下一个版本的设计中再进行修改。

设计签收需要所有当事人和部门的批准。

设计签收过程要用到以下六个指标，这些指标可通过第五章所描述的工具得到。

1. RTL 必须满足编码指导原则；
2. 设计必须满足功能覆盖和代码覆盖的目标；
3. FPGA 项目必须没有警告，若还有警告，则必须详细说明；
4. 必须满足设计说明书中的时序要求；
5. 必须满足系统在线调试的要求。在某些产品中，这可能包含老化测试和全环境测试；
6. 不满足设计说明书的每个情况都必须详细说明。

14.2　设计签收之后

在设计已经获准生产之后，需要把发布的版本和所有相关的设计和测试资料存档。这将作为设计所有未来版本的基础。

项目经理要主持项目完成后的总结会，讨论什么做对了、什么做错了以及从项目中学到了什么。这些信息将被用于未来项目的规划。

在设计发布会成功召开之后，可开始着手下一个项目的工作，它很可能就是该设计的下一个版本！

索　　引

国际视野　科技前沿

国际信息工程先进技术译丛

《FPGA —— 基于团队的最佳实践》
《基于片上去耦电容的配电网络》（原书第2版）
《智能摄像机》
《车载系统和安全的数字信号处理》
《嵌入式系统设计 —— 嵌入式信息物理系统基础》（原书第2版）
《纳米封装 —— 纳米技术与电子封装》
《内容分发网络》
《全面的功能验证：完整的工业流程》
《无线Mesh网络架构与协议》
《UMTS蜂窝系统的QoS与QoE管理》
《半导体制造与过程控制基础》
《WCDWA原理与开发设计》
《下一代移动系统：3G/B3G》
《IMS:IP多媒体概念和服务》（原书第2版）
《下一代无线系统与网络》
《深入浅出UMTS无线网络建模、
《规划与自动优化：理论与实践》
《HSDPA/HSUPA技术与系统设计——第三代移动
《通信系统宽带无线接入》
《无线传感器及元器件：网络、设计与应用》
《印制电路板——设计、制造、装配与测试》
《IPTV与网络视频：拓展广播电视的应用范围》
《多电压CMOS电路设计》
《微电子技术原理、设计与应用》
《蜂窝网络高级规划与优化2G/2.5G/3G/…向4G的演进》
《基于蜂窝系统的IMS——融合电信领域的VoIP演进》
《无线网络中的合作原理与应用》
《电生理学方法与仪器入门》
《移动电视：DVB-H、DMB、3G系统和富媒体应用》
《环境网络：支持下一代无线业务的多域协同网络》
《基于射频工程的UMTS空中接口设计与网络运行》
《未来UMTS的体系结构与业务平台：全IP的3G CDMA网络》
《UMTS-HSDPA系统的TCP性能》
《宽带无线通信中的空时编码》
《数字图像处理》（原书第4版）



上架指导　工业技术／电子技术／FPGA系统设计

ISBN 978-7-111-45264-5

定价：49.00元